PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 :		(11) International Publication Number: WO 92/15933
G06F 1/30, 3/00, 12/10 G06F 12/12, 13/00	A1	(43) International Publication Date: 17 September 1992 (17.09.92)

(21) International Application Number: PCT/US92/01845

(22) International Filing Date: 5 March 1992 (05.03.92)

(30) Priority data:

665,021	5 March 1991 (05.03.91)	US
860,731	21 February 1992 (21.02.92)	US

(71) Applicant: ZITEL CORPORATION [US/US]; 630 Alder Drive, Milpitas, CA 96035 (US).

(72) Inventor: LAUTZENHEISER, Marvin ; 2035 Lockwood Drive, San Jose, CA 95132 (US).

(74) Agents: CASERZA, Steven, F. et al.; Flehr, Hobach, Test, Albritton & Herbert, Four Embarcadero Center, Suite 3400, San Francisco, CA 94111-4187 (US).

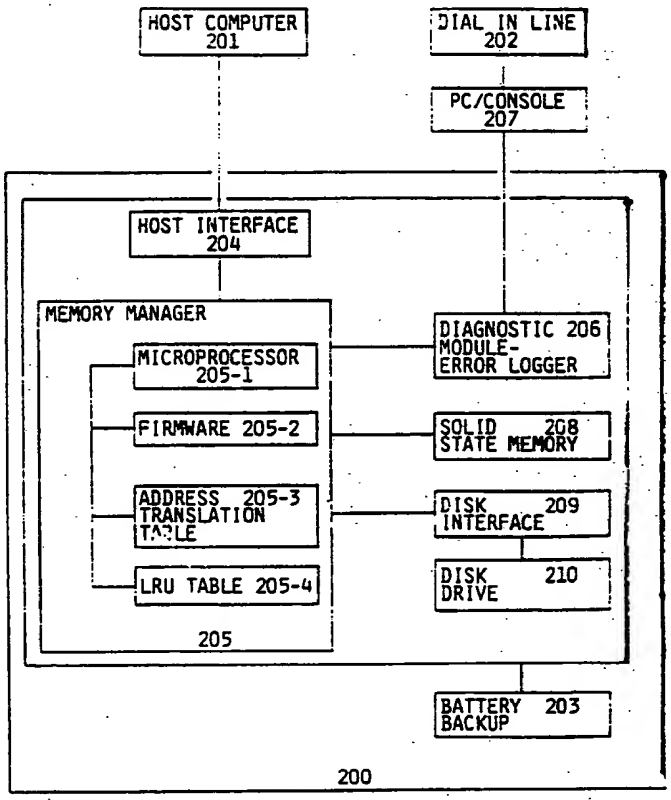
(81) Designated States: AT (European patent), AU, BE (European patent), CA, CH (European patent), DE (European patent), DK (European patent), ES (European patent), FR (European patent), GB (European patent), GR (European patent), IT (European patent), JP, KR, LU (European patent), MC (European patent), NL (European patent), SE (European patent).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: CACHE MEMORY SYSTEM AND METHOD OF OPERATING THE CACHE MEMORY SYSTEM

(57) Abstract

A data storage device (200) consists of solid state storage (208) and a rotating magnetic disk (210) attains a fast response time approaching that of a solid state device and improves response time of a normal magnetic disk. High performance is accomplished by hardware configuration (205) with procedures placing and maintaining data in the most appropriate media based on actual and projected activity. System management features a searchless method for determining the location of data within and between the two devices (208 and 210). Solid state memory (208) permits retention of useful, active data, and prefetching of data. Movement of updated data from the solid state storage (208) to the magnetic disk (210) and retrieval of prefetched data from the magnetic disk (210) is done as background tasks. The private channel between the solid state storage (208) and the disk (210) prevents the conflict : conversations. Microprocessors (205-1) manage : oversee the data transmission and storage. Data integrity is maintained through intelligent shutdown procedure.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria
AU	Australia
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CS	Czechoslovakia
DE	Germany
DK	Denmark
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IT	Italy
JP	Japan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
LI	Liechtenstein
LK	Sri Lanka
LU	Luxembourg
MC	Monaco
MG	Madagascar
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
NL	Netherlands
NO	Norway
PL	Poland
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SN	Senegal
SU	Soviet Union
TD	Chad
TG	Togo
US	United States of America

INTERNATIONAL SEARCH REPORT

International Application No.
PCT/ US92/01845

C .Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US, A, 4,468,730 (DODD ET AL.) 28 August 1984, See column 6, lines 8-26 and figures 1 and 2.	4-8
Y,E	US, A, 5,133,060 (WEBER ET AL.) 21 July 1992, See column 7, lines 1-43 and figure 1.	4-8
Y	US, A, 4,843,542 (DASHIELL ET AL.) 27 June 1989, See column 1, line 65 to column 2, line 6; column 4, lines 5-16 and figures 1, 2A and 2B.	9-15 and 25-28
Y	US, A, 4,888,691 (GEORGE ET AL.) 19 December 1989, See column 2, lines 33-37 and figure 19.	16-21,33 and 34

BOX II. OBSERVATIONS WHERE UNITY OF INVENTION WAS LACKING

This ISA found multiple inventions as follows:

GROUP I: Claim 1 drawn to a power control arrangement for a memory system classified in Class 395, Subclass 575.

GROUP II: Claims 2,3, 22-24 and 29-32 drawn to a method for operating a cache memory system using a Least Recently Used (LRU) protocol classified in Class 395, Subclass 425.

GROUP III: Claims 4-8 drawn to a method for operating a memory system including monitoring the number of available input/output operations classified in Class 395, Subclass 275.

GROUP IV: Claims 9-15 and 25-28 drawn to a cache memory system having a logical block address look-up table and address translation means classified in Class 395, Subclass 400.

GROUP V: Claims 16-21,33 and 34 drawn to a method of operating a cache system including writing data in selected tracks based on an amount of data written to the cache classified in Class 395, Subclass 425.

CACHE MEMORY SYSTEM AND METHOD OF OPERATING THE CACHE MEMORY SYSTEM

INTRODUCTION

Related Application

This application is a continuation-in-part application of U.S. Application Serial No. 07/665,021, filed March 5, 1991.

Field of Invention

This invention relates to a high performance computer data storage device including a combination of solid state storage and a rotating magnetic disk device.

Description of Prior Art

A number of computer data storage systems exist which make some use of solid state memory devices as a front end to rotating magnetic disk devices. A typical caching system uses a single solid state memory unit as a holding area for data stored on a string of magnetic disks, thereby allowing certain information be stored in a high speed cache memory, thereby increasing speed of performance as compared to the use solely of relatively lower speed disk memories, i.e. the percentage of times a desired piece of data is contained in the high speed cache memory, thereby allowing faster access as compared with when that data is only stored in a disk drive. A block diagram of such a system is shown in Figure 1. Host computer 101 communicates with the entire string of disks 102-1 through 102-N via cache unit 103 via Host interface 104, such as SCSI. All data going to or from disks 102 passes through the cache-to-disk data path consisting of host interface 104, cache unit 103, and disk interface 105. Cache unit 103 manages the caching of data and services requests from host computer 101. Major components of cache unit 103 include micro-processor 103-1, cache management hardware 103-2, cache

management firmware 103-3, address lookup table 103-4, and solid state cache memory 103-5.

The prior art cache system of Fig. 1 is intended to hold highly accessed data in a solid state memory area so as to give more rapid access to that data than would be achieved if the same data were accessed on the disk media. Typically, such cache systems are quite effective when attached to certain host computers and under certain workloads. However, there exist some drawbacks and, under certain conditions, such cache systems exhibit a performance level less than that achieved by similar, but uncached, devices. Some of the factors contributing to the less than desirable performance of prior art cached disk devices are now described.

The single cache memory 103-5 is used in conjunction with all disks in disk string 102. Data from any of the disks may reside in cache memory 103-5 at any given time. Busy data is favored for caching regardless of the disk drive on which it resides. When fulfilling a host command, the determination of whether or not the data is in cache memory 103-5, and the location of that data in cache memory 103-5, is usually via hashing schemes and table search operations. Hashing schemes and table searches can introduce time delays of their own which can defeat the purpose of the cache unit itself.

Performance is very sensitive to cache-hit rates. Due to caching overhead and queuing times, a low hit rate in a typical string oriented cache system can result in overall performance that is poorer than that of configured uncached string of disks.

The size of cache memory 103-5 relative to the capacity of disk drives 102 is generally low. An apparently obvious technique to remedy a low hit rate is to increase the cache memory 103-5 size. However, it has been found that there is an upper limit to the size of cache memory 103-5 above which adding more capacity has limited benefits. With limited cache memory 103-5

capacity, a multitude of requests over a variety of data segments exhausts the capability of the cache system to retain the desirable data in cache memory 103-5. Often, data that would be reused in the near future is decached prematurely to make room in cache memory 103-5 for handling new requests from the host computer 101. The result is a reduced cache hit rate. A reduced hit rate increases the number of disk accesses; increased disk accesses increases the contention on the data path. A self-defeating cycle is instituted.

"Background" cache-ahead operations are limited since the data transferred in such activities travels over the same data path as, and often conflicts with, the data transferred to service direct requests from the host computer 101. The data path between cache unit 103 and disk string 102 can easily be overloaded. All data to and from any of the disks in disk string 102, whether for satisfying requests from host computer 101 or for cache management purposes, travels across the cache-to-disk path. This creates a bottleneck if a large amount of prefetching of data from disk string 102 to cache memory 103-5 occurs. Each attempt to prefetch data from disk string 102 into cache memory 103-5 potentially creates contention for the path with data being communicated between any of the disk drives of disk string 102 and host computer 101. As a result, prefetching of data into cache memory 103-5 must be judiciously limited; increasing the size of the cache memory 103-5 beyond a certain limit does not produce corresponding improvements in the performance of the cache system. This initiates a string of related phenomena. Cache-ahead management is often limited to reading ahead on each host read command which is not fulfilled from the cached data. This technique helps to minimize the tendency of cache-ahead to increase the queuing of requests waiting for the path between cache memory 103-5 and disk string 102. However, one of the concepts on which caching is based is that data accesses

tend to be concentrated within a given locality within a reasonably short time frame. For example, data segments are often accessed in sequential fashion. Limiting the cache-ahead operations to being a function of read misses can have the negative effect of lowering the cache hit rate since such limitation may prevent or degrade the exploitation of the locality of data accesses.

A variety of algorithms and configurations have been devised in attempts to optimize the performance of string caches. A nearly universally accepted concept involves the retention and replacement of cached data segments based on least-recently used (LRU) measurements. The decaching of data to make room for new data is managed by a table which gives, for each cached block of data, its relative time since it was last accessed. Depending on the algorithm used, this process can also result in some form of table search with a potential measurable time delay.

Cache memory 103-5 is generally volatile; the data is lost if power to the unit is removed. This characteristic, coupled with the possibility of unexpected power outages, has generally imposed a write-through design for handling data transferred from host computer 103 to the cached string. In such a design, all writes from host computer 103 are written directly to disk; handled at disk speed, these operations are subject to all the inherent time delays of seek, latency, and lower transfer rates commonly associated with disk operations.

Cache unit 103 communicates with the string of disk drives 102 through disk interface 105.

SUMMARY OF THE INVENTION

Computer operations and throughput are often limited by the time required to write data to, or read data from, a peripheral data storage device. A solid state storage device has high-speed response, but at a

relatively high cost per megabyte of storage. A rotating magnetic disk, optical disk, or other mass media provides high storage capacity at a relatively low cost per megabyte, but with a low-speed response. The teachings of this invention provides a hybrid solid state and mass storage device which gives near solid state speed at a cost per megabyte approaching that of the mass storage device. For the purposes of this discussion, embodiments will be described with regard to magnetic disk media. However, it is to be understood that the teachings of this invention are equally applicable to other types of mass storage devices, including optical disk devices, and the like.

This invention is based on several features: a rotating magnetic disk media, an ample solid state storage capacity, a private channel between the disk and solid state storage devices, microprocessors which perform unique data management, a unique prefetch procedure, and parallel activity capabilities. The hybrid storage media of this invention performs at near solid state speeds for many types of computer workloads while practically never performing at less than normal magnetic disk speeds for any workload.

A rotating magnetic disk media is used to give the device a large capacity; the solid state storage is used to give the device a high-speed response capability. By associating the solid state media directly with a single magnetic disk device, a private data communication line is established which avoids contention between normal data transfers between the host and the device and transfers between the solid state memory and the disk. This private data channel permits virtually unlimited conversation between the two storage media. Utilization of ample solid state memory permits efficient maintenance of data for multiple, simultaneously active data streams. Management of the storage is via one or more microprocessors which utilize historical and projected data accesses to perform

intelligent placement of data. No table searches are employed in the time-critical path. Host accesses to data stored in the solid state memory are at solid state speeds; host accesses to data stored on the magnetic disk are at disk device speeds. Under most conditions, all data sent from the host to the device is handled at solid state speeds.

Intelligence is embodied to cause the device to shift itself into a pseudo magnetic disk mode when workloads and the characteristics of the current I/O indicate that such mode is the optimal mode of operation. This shift of mode assists in maintaining performance at a high level. Additionally, based on historical information, the embodied intelligence may shift operating priorities within the device in order to optimize the handling of near-future host-device I/O operations.

BRIEF DESCRIPTIONS OF THE DRAWINGS

Figure 1 is a block diagram of a typical prior art cached disk computer data storage system;

Figure 2 is a block diagram depicting one embodiment of a cached disk computer data storage device constructed in accordance with the teachings of this invention;

Figure 3 is a block diagram depicting one embodiment of a hardware configuration which implements the described invention;

Figure 4 is a flow chart depicting the operation of one embodiment of this invention;

Figure 5 is a flow chart depicting a more detailed description of the operation of the host command step of Figure 4;

Figure 6 is a flow chart depicting the operation of one embodiment of the analyzed host I/O command operation of Figure 5;

Figure 7 is a flow chart depicting in more detail the operation of the setup track address list operation of Figure 6;

Figure 8 is a flow chart depicting in more detail the address translation of Figure 7;

Figure 9 is a flow chart depicting the cache read hit operation depicted in Figure 5;

Figure 10 is a flow chart depicting in more detail the cache read miss operation depicted in Figure 5;

Figure 11 is a flow chart depicting the cache write hit operation of Figure 5;

Figure 12 is a flow chart depicting the cache write miss operation of Figure 5;

Figure 13 is a flow chart depicting the seek cache miss operation of Figure 5;

Figure 14 is a flow chart depicting the decache LRU operation of Figures 6, 13 and 15;

Figure 15 is a flow chart depicting the cache ahead operation of Figure 4;

Figure 15 is a flow chart depicting the operation of the cache ahead determination operation of Figure 15;

Figure 17 is a flow chart depicting the operation of the initiate background sweep operation of Figure 4;

Figure 18 is a flow chart depicting the step of background sweep initiation at host I/O completion depicted in Figure 4;

Figure 19 is a flow chart depicting the generate background event operations depicted in Figures 17, 18, and 20;

Figure 20 is a flow chart depicting the operation of the continued background sweep step of Figure 4;

Figure 21 is a flow chart depicting the power down control operations; and

Figure 22 is a flow chart depicting the final background sweep operation depicted in Figure 21.

Description of the Tables

Tables F-1 through F-4 describe the organization of Tables T-1 through T-4, respectively;

Table T-1 depicts an example of values in the address translation (ADT) table prior to the handling of the first I/O operation from the host CPU;

Table T-2 depicts an example of values in the Least-Recently-Used (LRU) table prior to the handling of the first I/O operation from the host CPU;

Table T-3 describes the least recently used (LRU) table;

Tables T-3a through T-3e depict the ADT table after various numbers of I/O operations; and

Table 4 depicts a sample of I/O commands extracted from computer system operations during normal usage. These I/O commands, and the intervening commands, were the basis for the sample predicted LRU and ADT tables as shown in Tables T-1 through T-3.

DETAILED DESCRIPTION OF THE INVENTION

Glossary of Terms

ADDRESS TRANSLATION: The conversion of a sector address into a track address and sector offset within the track.

CACHE-AHEAD FACTOR; PROXIMITY FACTOR: At each track hit or rehit, cached data sufficient to satisfy a number of I/O's may remain in front of, and/or behind, the current location of the data involved in the current I/O. When either of these two remaining areas provide for less than a set number of I/O's, the cache-ahead is activated. That minimum number of potential I/O's is the cache-ahead factor, or the proximity factor.

ADDRESS TRANSLATION TABLE; ADT TABLE: The table which maintains the relationship between disk track

identifiers and solid state memory addresses; also holds other information as required.

CACHE: The solid state memory area which holds user data within the unit, taken as a whole.

CPU SECTOR: See Logical Sector.

DISK; MAGNETIC DISK; ROTATING MAGNETIC DISK: A rotating magnetic media disk drive.

DISK SECTOR ADDRESS: The address of a physical sector on the magnetic disk unit.

DISK SERVER: The logical section of the unit which handles the writes to, and reads from, the rotating magnetic disk.

DISK TRACK ADDRESS; TRACK ADDRESS: The address of the first sector of data in a given track on disk. These addresses correspond to physical locations on the rotating magnetic disk. Each sector address as specified in an I/O operation can be converted into a track address and a sector offset within that track.

DMA: Direct Memory Access; that is, memory-to-memory transfer without the involvement of the processor.

DRAM: Dynamic random access memory. The chip or chips that are used for solid state memory devices.
EDAC: Error Detection And Correction

EEPROM: Electrically Erasable Programmable Read-Only Memory

EPROM: Erasable Programmable Read-Only Memory

HOST: The computer to which the unit is attached.

HOST SERVER: The portion of the unit which interfaces with the host computer.

I/O SIZE: The size of a host I/O request as a number of sectors.

LRU: Least-Recently-Used, as pertains to that data storage track which has not been accessed for the longest period of time.

LRU TABLE; LEAST-RECENTLY-USED TABLE: The table containing the information which allows the unit's controller to determine which solid state memory data areas may be reused with the least impact on the cache efficiency.

MRU: Most-Recently-Used, as pertains to that data storage track which has been accessed in the nearest time past.

NORMAL MODE: The condition of the device in which it can use its normal priorities in order to reach its optimal performance level.

NULL, NULL VALUE: A value in a table field which indicates the field should be considered to be empty; depending on usage, will be zero, or will be the highest value the bit structure of the field can accommodate.

PHYSICAL TRACK; DISK TRACK: A complete data track on disk; one complete magnetic band on one platter of the disk unit.

PROXIMITY FACTOR: At each track hit or rehit, data sufficient to satisfy a number of I/O's may remain in front of, and/or behind, the current location of the data involved in the current I/O. When either of these

two remaining areas provide for less than a set number of I/O's, the cache-ahead is activated. That minimum number of potential I/O's is the cache-ahead factor, or the proximity factor.

READ-MISS-MAXSIZE: The size of a host read transaction as a number of sectors which, when exceeded, causes the transaction to be handled in pseudo disk mode.

RECYCLE: The term used to describe the retention of a track in cache beyond its arrival at the LRU position; such retention to be based on the fact the track was reused at some time since it staged into cache as a result of a cache-ahead operation, or since it was last the beneficiary of a recycling action.

SCSI: Small Computer System Interface; the name applied to the protocol for interfacing devices, such as a disk device to a host computer.

SCSI CONTROL CHANNEL: A physical connection between devices which uses the SCSI protocol, and is made up of logical controllers connected by a cable.

SECTOR: The logical sub-unit of a disk track; the smallest addressable unit of data on a disk.

SOLID STATE MEMORY, SOLID STATE DEVICE; SSD: Storage media made up of solid state devices such as DRAMs.

SSD TRACK ADDRESS: The address in the solid state memory at which the first byte of the first sector of a given disk track resides.

TRACK; LOGICAL TRACK; LOGICAL BLOCK: A logical data track on disk, or its equivalent in SSD; may or may not be identical to a physical track on disk (one complete

magnetic band on one platter of the disk unit). It is noted that an I/O operation may involve more than one logical block.

TRACK SIZE: The number of sectors considered to be in a disk track; this may or may not be equal to the actual number of sectors in a disk track.

UNIT: The designation of the unit being described herein.

URGENT MODE: The condition of the device in which it must shift priorities in order to maintain at least magnetic disk level performance.

WRITE-MISS-MAXSIZE: The size of a host write transaction as a number of sectors which, when exceeded, causes the transaction to be handled in pseudo disk mode.

System Overview

In accordance with the teachings of this invention, a computer peripheral data storage device is provided comprising a combination solid state memory and rotating magnetic disk; such device having the large capacity of magnetic disk with near solid state speed at a cost per megabyte approaching that of magnetic disk media. For the purposes of this discussion, embodiments will be described with regard to magnetic disk media. However, it is to be understood that the teachings of this invention are equally applicable to other types of mass storage devices, including optical disk devices, and the like.

This device of invention derives its large storage capacity from the rotating magnetic disk media. Its high speed performance stems from the combination of a private channel between the two storage media, multiple microprocessors utilizing a set of unique data

management algorithms, a unique prefetch procedure, parallel activity capabilities, and an ample solid state memory. This hybrid storage media gives overall performance near that of solid state memory for most types of computer workloads while practically never performing at less than normal magnetic disk speeds for any workload.

To the host computer, the device of this invention appears to be a single, directly addressable entity. By the combination, within the device, of a solid state memory and one or more magnetic disk devices, private data communication lines are established within the device which avoids contention between normal data transfers between the host and the device, and transfers between the solid state memory and the disk media. This private data channel permits unrestricted data transfers between the two storage media with practically no contention with the communication between the host computer and the described device. Utilization of ample solid state memory permits efficient retention of data for multiple, simultaneously active data streams. Management of the storage is via microprocessors which anticipate data accesses based on historical activity. Data is moved into the solid state memory from the disk media based on management algorithms which insure that no table searches need be employed in the time-critical path. Host computer accesses to data stored in the solid state memory are at near solid state speeds; accesses to data stored on the magnetic disk are at near disk device speeds. All data sent from the host to the device is transferred at solid state speeds limited only by the channel capability.

Hardware Description

A device constructed in accordance with the teachings of this invention is depicted in Figure 2. Memory device 200 is a self-contained module which includes three points of contact with the outside world.

Its primary contact is with host computer 201 via host interface 204. Host interface 204 comprises, for example, a dedicated SCSI control processor which handles communications between host computer 201 and memory manager 205. An operator interface is provided via the console 207, which allows the user to exercise overall control of the memory device 200. Another point of contact is a dial-in line or PC computer connection for interrogating the unit's status or operating condition of memory device 200.

Memory manager 205 handles all functions necessary to manage the storage of data in, and retrieval of data from disk drives 210 (or high capacity memory devices) and solid state memory 208, the two storage media. The memory manager 205 consists of one or more microprocessors 205-1 or the like, associated firmware 205-2, and management tables, such as Address Translation (ADT) Table 205-3 and Least Recently Used (LRU) Table 205-4.

Solid state memory 208 is utilized for that data which memory manager 205, based on its experience, deems most useful to host computer 201, or most likely to become useful in the near future.

Magnetic disk 201 is the ultimate storage for all data, and provides the needed large storage capacity. Disk interface 209 serves as a separate dedicated control processor (such as an SCSI processor) for handling communications between memory manager 205 and disk drive 210.

Information about functional errors and operational statistics are maintained by diagnostic module-error logger 206. Access to module 206 is by either dial-in line 202 or console 207. Console 207 serves as the operator's access to the memory device 200 for such actions as powering the unit on and off, reading or resetting the error logger, or inquiring of the unit's statistics.

The memory device 200 includes power backup system 203 which includes a rechargeable battery. Backup system 203 is prepared to maintain power to memory device 200 should normal power be interrupted. If such a power interruption occurs, the memory manager 205 takes whatever action is necessary to place all updated data stored in solid state memory 208 onto magnetic disk 210 before shutting down memory device 200.

Figure 3 depicts a hardware controller block diagram of one embodiment of this invention. As shown in Figure 3, hardware controller 300 provides three I/O ports, 301, 302, and 303. I/O ports 301 and 302 are differential SCSI ports used to connect hardware controller 300 to one or more host computers 201 (Figure 2). I/O port 303 is a single-ended SCSI port used to connect controller 300 to disk drive 210 (which in this embodiment is a 5.25" magnetic hard disk drive). Disk drive 210 provides long-term non-volatile storage for data that flows into controller 300 from host computers 201. "Differential" and "single-ended" refer to specific electrical characteristics of SCSI ports; the most significant distinction between the two lies in the area of acceptable I/O cable length. The SCSI aspects of I/O ports 301, 302, and 303 are otherwise identical.

Cache memory 308 (corresponding to memory 208) is a large, high-speed memory used to store, on a dynamic basis, the currently active and potentially active data. The storage capacity of cache memory 308 can be selected at any convenient size and, in the embodiment depicted in Figure 3, comprises 64 Megabytes of storage. Cache memory 308 is organized as 16 Megawords; each word consists of four data bytes (32 bits) and seven bits of error-correcting code. Typically, the storage capacity of cache memory 308 is selected to be within the range of approximately one-half of one percent to 100 percent of the storage capacity of the one or more magnetic disks 210 (Figure 2) with which it operates. A small portion of cache memory 308 is used to store the tables

required to manage the caching operations; alternatively, a different memory (not shown, but accessible by microcontroller 305) is used for this purpose.

EDAC circuitry 306 performs error detecting and correcting functions for cache memory 308. In this embodiment, EDAC circuitry 306 generates a seven-bit error-correcting code for each 32-bit data word written to cache memory 308; this information is written to cache memory 308 along with the data word from which it was generated. The error-correcting code is examined by EDAC circuitry 306 when data is retrieved from cache memory 308 to verify that the data has not been corrupted since last written to cache memory 308. The modified hamming code chosen for this embodiment allows EDAC circuitry 306 to correct all single-bit errors that occur and detect all double-bit and many multiple-bit errors that occur.

Error logger 307 is used to provide a record of errors that are detected by EDAC circuitry 306. The information recorded by error logger 307 is retrieved by microcontroller 305 for analysis and/or display. This information is sufficiently detailed to permit identification by microcontroller 305 of the specific bit in error (for single-bit errors) or the specific word in error (for double-bit errors). In the event that EDAC circuitry 306 detects a single-bit error, the bit in error is corrected as the data is transferred to whichever interface requested the data (processor/cache interface logic 311, host/cache interface logic 311 and 312, and disk/cache interface logic 313). A signal is also sent to microcontroller 305 to permit handling of this error condition (which involves analyzing the error based on the contents of error logger 307, attempting to scrub the error, and analyzing the results of the scrub to determine if the error was soft or hard).

In the event that EDAC circuitry 306 detects a double-bit error, a signal is sent to microcontroller

305. Microcontroller 305 will recognize that some data has been corrupted. If the corruption has occurred in the ADT or LRU tables, an attempt is made to reconstruct the now-defective table from the other, then relocate both tables to a different portion of cache memory 308.

If the corruption has occurred in an area of cache memory 308 that holds user data, microcontroller 305 attempts to salvage as much data as possible (transferring appropriate portions of cache memory 308 to disk drive 210, for example) before refusing to accept new data transfer commands. Any response to a request for status from the host computer 201 will contain information that the host computer 201 may use to recognize that memory device 200 is no longer operating properly.

Microcontroller 305 includes programmable control processor 314 (for example, an 80C196 microcontroller available from Intel Corporation), 64 kilobytes of EPROM memory 315, and hardware to allow programmable control processor 314 to control the following: I/O ports 301, 302, and 303, cache memory 308, EDAC 306, error logger 307, host/cache interface logic 311 and 312, disk/cache interface logic 313, processor/cache interface logic 316, and serial port 309.

Programmable control processor 314 performs the functions dictated by software programs that have been converted into a form that it can execute directly. These software programs are stored in EPROM memory 315.

In one embodiment, the host/cache interface logic sections 311 and 312 are essentially identical. Each host/cache interface logic section contains the DMA, byte/word, word/byte, and address register hardware that is required for the corresponding I/O port (301 for 311, 302 for 312) to gain access to cache memory 308. Each host/cache interface logic section also contains hardware to permit control via microcontroller 305. In this embodiment I/O ports 301 and 302 have data path

widths of eight bits (byte). Cache memory 308 has a data path width of 32 bits (word).

Disk/cache interface logic 313 is similar to host/cache interface logic sections 311 and 312. It contains the DMA, byte/word, word/byte, and address register hardware that is required for disk I/O port 303 to gain access to cache memory 308. Disk/cache interface logic 313 also contains hardware to permit control via microcontroller 305. In this embodiment, I/O port 303 has a data path width of eight bits (byte).

Processor/cache interface logic 316 is similar to host/cache interface logic sections 311 and 312 and disk/cache interface logic 313. It contains the DMA, half-word/word, word/half-word, and address register hardware that is required for programmable control processor 314 to gain access to cache memory 308. Processor/cache interface logic 316 also contains hardware to permit control via microcontroller 305. In this embodiment, programmable control processor 314 has a data path width of 16 bits (half-word).

Serial port 309 allows the connection of an external device (for example, a small computer) to provide a human interface to the system 200. Serial port 309 permits initiation of diagnostics, reporting of diagnostic results, setup of system 200 operating parameters, monitoring of system 200 performance, and reviewing errors recorded inside system 200. In other embodiments, serial port 309 allows the transfer of different and/or improved software programs from the external device to the control program storage (when memory 315 is implemented with EEPROM rather than EPROM, for example).

Formats of control tables

Format of Address Translation (ADT) Table

The Address Translation Table, along with the LRU table, maintains the information required to manage the caching operations. There are two sections in the ADT table, the indexed, or tabular portion, and the set of unindexed, or single-valued items.

The unindexed portion of the ADT table contains two types of data fields; the first are those items which are essential to the cache management, the second category contains those data items which maintain records of the unit's performance.

The first group of unindexed items, or those requisite to the cache management, includes the following single-valued items.

1) ADT-CNL. The number of tracks on the cached disk spindle; also equals the number of lines in the ADT table. This is set at the time the Ready cache unit is configured and is not changed while the unit is in operation.

2) ADT-HEAD-POS. The current position of the read/write head of the cache disk. This is updated every time the head is positioned.

3) ADT-SWEEP-DIR. The direction in which the current sweep of the background writes is progressing. This is updated each time the sweep reverses its direction across the disk.

4) ADT-MOD-COUNT. The total number of tracks in the cache which have been modified by writes from the host and are currently awaiting a write to disk by the Disk server. This is increased by one whenever an unmodified cache track is updated by the host, and it is decreased by one whenever a modified cache track is copied to the cache disk.

The second group of unindexed items are those which record the unit's performance, and are all used to compute the current operating characteristics of the unit. They include the following single-valued items.

1) ADT-READ-HITS. The number of cache read-hits encountered since the last reset. This value is set to zero by a reset operation from the PC/console. It is incremented by one for each read I/O which is entirely satisfied from data which is resident in the cache memory.

2) ADT-READ-MISSES. The number of cache read-misses encountered since the last reset. This value is set to zero by a reset operation from the PC/console. It is incremented by one for each read I/O which cannot be entirely satisfied from data which is resident in the cache memory.

3) ADT-WRITE-HITS. The number of cache write-hits encountered since the last reset. This value is set to zero by a reset operation from the PC/console. It is incremented by one for each write I/O for which the corresponding track or tracks are found to be in cache memory.

4) ADT-WRITE-MISSES. The number of write-misses encountered since the last reset. This value is set to zero by a reset operation from the PC/console. It is incremented by one for each write I/O for which one or more of the corresponding track or tracks are not found to be in cache memory.

There is one line in the tabular portion for each data track on the spindle. A line is referred to by its line number, or index. That line number directly corresponds to a track number on the disk. When the

host wants to access or modify data on the disk, it does so by referencing a starting sector address and indicating the number of sectors to be accessed or modified. For caching purposes, the starting sector address is converted into a track identifier and offset within that track.

A disk sector address is converted into a track number and a sector offset by dividing it by the number of sectors per track. The remainder is the offset into the track. The quotient is the track identifier and is the index into the ADT table. Using this index, the condition of the specified disk track can be determined directly from data in the ADT table; no search is required to determine cache-hits or misses.

Each ADT line contains the following items:

1) ADT-SLOT. The number of the cache slot which contains the data which is also held in the disk track of the same number as the index of this ADT table line. By design, the value in ADT-SLOT also points to the line in the LRU table related to the cached disk track. If the disk track is not in cache, the value in this field is meaningless and is set to its null value. It is by means of this field that cache-hits can be serviced completely without any table search. A null value in this field indicates the corresponding disk track is not stored in the cache. This field is updated each time a track is entered into or removed from the SSD area.

2) ADT-MODIFIED. A flag indicating whether or not the corresponding cached track has been modified by a write operation from the host, and thus, needs copied from the cache to the disk.

Format of Least Recently Used (LRU) Table

The LRU table maintains the information relative to the times when cached tracks of data were last accessed. This information is necessary for the unit to always be aware of which cache slots are available for overwriting whenever uncached data tracks must be placed in cache. Its contents also provide redundancy for the data kept in the ADT table, thus contributing to system reliability.

There are two sections in the LRU table, the indexed, or tabular portion, and the set of unindexed, or single- valued items. The unindexed portion of the LRU table contains data required to manage the caching process. The tabular portion is composed of pointers for LRU chaining purposes, pointers into the ADT table, and the recycle control markers.

It is by means of this LRU information and the ADT table information that the system determines which cached track to overwrite when a cache area is needed for an uncached disk track. The unindexed items are requisite to the cache management, and includes the following single-valued items.

1) LRU-CNL. The number of track-equivalent slots in the cache area; this is equal to the number of lines in the LRU table.

2) LRU-LRU. The LRU-LRU table element points to the cache area track-slot containing the cached data which has been left untouched for the longest time. It is updated when new activity for the referenced slot makes it no longer the least-recently- used. The referenced slot is the top candidate for overwriting when new data must be written into the cache.

3) LRU-MRU. The LRU-MRU table element points to the cache area track-slot containing the cached data which

has been most- recently referenced by the host. LRU-MRU is updated every time a track is touched by either a read or a write from the host. At that time, the address of the accessed track is placed in LRU-MRU and the LRU chains are updated in the indexed portion of the LRU table.

There is one line in the tabular portion for each datatrack- slot in the cache data area. A line is referred to by its line number, or index. That line number directly corresponds to a slot in the cache data area.

Each LRU table line contains pointer fields plus other control fields.

1) LRU-TRACK. The pointer to the ADT line which references the disk track currently resident in the corresponding cache slot. By design, this value is also the identifier of the disk track whose data currently resides in the corresponding cache slot, if any.

2) LRU-LAST. This is part of the bidirectional chaining of the cache data slots. LRU-LAST is the pointer to the next-older (in usage) cache slot. If this slot is the oldest, LRU-LAST will contain a zero.

3) LRU-NEXT. This is the other half of the bidirectional chaining of the cache data slots. LRU-NEXT is the pointer to the next newer (in usage) cache slot. If this slot is the newest, LRU-NEXT will contain a zero.

4) LRU-CACHED-LOW. A field containing the track-relative number of the lowest sector of this cached track which contains valid cached data.

5) LRU-CACHED-HIGH. A field containing the track-relative number of the highest sector of this cached track which contains valid cached data.

6) LRU-MOD-LOW. A field containing the track-relative number of the lowest sector of this cached track which contains modified cached data.

7) LRU-MOD-HIGH. A field containing the track-relative number of the highest sector of this cached track which contains modified cached data.

8) LRU-LOCKED. A flag indicating whether or not the corresponding cached track is currently the target of some operation, such as being acquired from the disk, being modified by the host, or being written to the disk by the cache controller; such activity making the track unavailable for certain other operations.

9) LRU-RECYCLE. A single bit marker which indicates whether or not the corresponding track is a candidate for recycling. It is set to 1 (on) whenever the track is reused; it is set to 0 (off) when the track is recycled (moved to the MRU position). It is initially set to 0 when a track is brought into cache as a result of a cache-ahead decision. For a track brought in to satisfy a cache miss, it is set to 1.

Examples of Tables

Initial ADT table

When a unit is first powered on, the ADT table is in an indeterminate state. In order to become operational, initial values must be entered into their appropriate table elements. Initial values for unindexed fields of the ADT table are as follows:

The ADT-CNL field must be set to the size of the cache disk as a number of tracks.

The ADT-HEAD-POS field is set to zero to indicate the head is currently at the edge of the disk. This may, or may not, be true, but it does not matter; it will become correct on the first access to the disk.

The ADT-SWEEP-DIR field is arbitrarily set to one to indicate the head is moving in an upward (based on track addresses) direction. This will be corrected at the initiation of the first background sweep.

The ADT-MOD-COUNT field is set to zero to reflect the fact that no modified tracks are waiting in cache to be copied to disk.

The ADT-READ-HITS field is set to zero to reflect the fact that no cache hits have occurred during read operations.

The ADT-READ-MISSES field is set to zero to reflect the fact that no cache misses have occurred during read operations.

The ADT-WRITE-HITS field is set to zero to reflect the fact that no cache hits have occurred during write operations.

The ADT-WRITE-MISSES field is set to zero to reflect the fact that no cache misses have occurred during write operations.

All indexed fields of all lines of the ADT table are initially set to zero to indicate that no tracks are resident in cache.

Initial LRU table

When a unit is first turned on, the LRU table is in an indeterminate state. In order to become operational, initial values must be entered into their appropriate table elements. While there are many acceptable ways to initialize the chaining fields, a simple one has been selected, and is described here.

Initial values for unindexed fields of the LRU table are as follows:

The LRU-CNL field must be set to the size of the cache, as a number of track-equivalent slots.

The LRU-LRU field is set to one to represent the lowest numbered cache slot as being the oldest. This is an arbitrary choice in keeping with the chaining values selected, below.

The LRU-MRU field is set equal to LRU-CNL to represent the highest cache slot as being the most recently used. This is an arbitrary choice in keeping with the initial chaining values selected, below.

Initial values for indexed fields of the LRU table are as follows:

The LRU-TRACK field of every line of the LRU table is set to zero to indicate that no disk data tracks are currently held in cache.

The LRU-LAST field of every line of the LRU table is set to that line's index minus one. This action, along with the settings for the LRU-NEXT values, produces a chained list suitable for the cache start-up operation.

The LRU-NEXT field of every line, except the highest, of the LRU table is set to that line's index plus one. The LRU-NEXT field of the highest line is set to zero. These settings, along with the settings for the LRU-LAST values, produces a chained list suitable for the cache start-up operation.

The LRU-CACHED-LOW field of every line is set to its null value to indicate that no portion of the disk track is currently held in cache.

The LRU-CACHED-HIGH field of every line is set to its null value to indicate that no portion of the disk track is currently held in cache.

The LRU-MOD-LOW field of every line is set to its null value to indicate that no portion of the disk track currently held in cache is in a modified condition.

The LRU-MOD-HIGH field of every line is set to its null value to indicate that no portion of the disk track currently held in cache is in a modified condition.

The LRU-LOCKED field of every line of the LRU table is set to zero to indicate no cache slot is currently locked.

The LRU-RECYCLE field of every line of the LRU table is set to zero to indicate that no slot is currently a candidate for recycling.

Operational state ADT tables

The operational state ADT table examples illustrate the conditions after the very first (sample) I/O has occurred and after the system has reached a fully active condition. These fully active examples show the effects of several I/O's on the state of the ADT table contents. Also included in these examples is the effect of a background sweep which wrote modified tracks from cache to disk. A detailed description of these specific sample operations appears under the LRU table discussion, below.

Operational state LRU tables

The operational state LRU table examples illustrate the conditions after the very first I/O has occurred and after the system has reached a fully active condition. These fully active examples show the effects of several I/O's on the state of the LRU table contents.

Description of Sample I/O's

For purposes of illustration, a sample of I/O's were chosen to be discussed in detail. Those chosen for discussion are I/O numbers 1, 1000, 1001, and 1002 taken from a trace of actions at an operational site; they are detailed in Table 4 as projected into the described system. The following discussions of the sample I/O's include the effects on both the ADT and the LRU tables.

Actions related to I/O operation number 1:

1. This I/O is a read involving disk track numbers 46 and 47; since nothing is in cache, it must be a cache-miss. A portion of track 46, and all of track 47

is brought into cache. The ADT table is modified to show the locations of the tracks in cache; the amount of each track now cached is recorded; the chains are relinked to show the tracks to be the MRU and the next-to-MRU tracks; and they are both marked for recycling. They are marked for recycling since they will have been used at least once by the time they reach the LRU position in the chain.

2. Based on the I/O size and the distance from the end of the I/O in track 47, a prefetch of track 48 is initiated. That activity is not reflected in the ADT and LRU tables since it is initiated as a background operation after the completion of the current I/O.

After 999 I/O's have occurred, the ADT and LRU tables have reached a certain status. I/O number 1000 is a read of 68 sectors starting at sector address 14,190. This occupies sectors 11 through 78 of disk track 56. Based on these conditions, the following actions relate to I/O operation number 1000:

1. This I/O is a read involving track 56 which is not in cache; the required portion of it must be brought into cache. While fetching this required data, the portion of track 56 from the end of the requested data to the end of the track is also fetched. This is done here since this is the most expeditious time to do so, and satisfies the prefetch rules. The LRU table is updated to reflect the caching of track 56 and slot into which this data is fetched is placed at the MRU position in the LRU chain.

2. To make room for caching track 56, the old LRU track was decached.

3. A read I/O operation does not affect the need for a background sweep. There are three tracks in cache that

need to be copied to disk; this condition remains unchanged by the current I/O.

4. Several cache slots have been marked for recycling, including the slot at the LRU end of the chain and the slot it points to. Before any prefetch is initiated, these will be moved to the MRU and next-to-MRU positions and their recycle markers will have been removed.

5. Since track 57 is already in cache, no prefetch is needed for it.

5. The size of the I/O (68 sectors) and the current I/O's proximity to the first sector in track 56 indicate that track 55 should be prefetched by a cache-ahead action. That prefetch will be initiated as a background operation. Based on the LRU chain and the recycling situation, track 55 will be cached into slot 3. For the moment it will occupy the MRU position in the chain.

Actions related to I/O operation number 1001:

1. Prefetching of track 55, which was initiated by I/O 1000 has now been completed.

2. I/O number 1001 is a write involving sectors 191-256 of track 61, and sectors 1-2 of track 62. The LRU and ADT table references to them are updated.

3. This I/O action modified two cached tracks, bringing the total number of tracks which need written to disk up to the trigger point for the background sweep.

4. The background sweep is initiated and starts writing the modified tracks from cache to disk.

5. Since the background sweep is using the disk spindle, no cache-ahead is initiated, even though the unit would consider the prefetch of track 60 into cache.

Actions related to I/O operation number 1002:

1. The background sweep completed writing all modified tracks from cache to disk; it then went into the dormant state.

2. I/O 1002 was a write involving track 214 which is already resident in cache. The track is marked in the ADT table as having been modified. In the LRU table, track 214 in slot number 13 is removed from the MRU-LRU chain, and the amount modified is recorded in the LRU-MOD-LOW and LRU-MOD-HIGH fields.

3. A prefetch of track 215 is initiated since the position of the current I/O in track 214 is near enough to the end of the track to warrant a cache-ahead operation. This activity does not appear in the ADT and LRU tables for I/O 1002 since it will occur in the background after the completion of the current I/O.

4. Since a prefetch of track 215 has been initiated, track 213 is not considered for prefetching.

FIRMWARE

Firmware Overview

The memory controller of this invention goes into a semi-dormant state when there is no activity that it needs to handle. As depicted in the flow chart of Figure 4, there are three types of occurrences that may cause the controller to become active:

1. The host computer sends a command;
2. The background sweep completes an event;
3. The background sweep times out.

Insofar as possible, the host computer commands are given priority over other memory device activities. Thus, when a command is received from the host, it is immediately turned over to the Host Command Handler (described elsewhere). At the completion of the activity called for by that command, the memory controller determines if the background sweep is active. If it is not active, the background status is inspected and action is taken as appropriate, as described later with regard to the background check. Following the background status check, the cache-ahead status is checked, as described later with regard to the cache-ahead status. The controller then waits for the next host command. The controller may not be completely inactive at this time, inasmuch as either the background sweep or the cache-ahead may have initiated or continued some disk activity. If the background was found to be active, its activity is continued until such time as it has no more immediate work to do, as described later with regard to background continuation.

When the background sweep completes a command, the controller is given an interrupt with a signal that indicates the sweep needs its attention. At that time, the controller initiates the next sweep event, if any is waiting, and sets up a follow on sweep event, also based on need, as described later with regard to the background continuation. At the completion of each sweep event, the controller determines if there is a need to continue the sweep. If no such need exists, the background sweep is placed in the dormant state. In either case, when the controller has completed its housekeeping, it becomes inactive awaiting its next task.

The background sweep can be activated in either of two ways; it will be activated when a set number of cached tracks have been modified and are in need of

being written from cache to disk. The sweep may also be activated by a timeout. A timeout occurs whenever the sweep is inactive, and there exists any modified track waiting to be written from cache to disk which has been waiting more than a preset amount of time. When a timeout occurs, the controller is signaled that the sweep requires its attention. The controller initiates the background sweep (see description of background initiation) and, after completing the appropriate housekeeping, awaits the next command or event requiring its attention. The background sweep itself continues in operation until there is no more immediate need for its services. At that time it also is returned to the dormant state.

Host-Command Management

Whenever a command is received from the host computer, it is given the highest possible priority and handled as depicted in Figure 5. To determine what actions are required, the command must be analyzed. A portion of the firmware is dedicated to that purpose (see description of host command analysis). The analysis of the command determines the type of command (read, write, seek, or other) and, where meaningful, will make a cache hit/miss determination. The analysis also sets up a table of one or more lines which will be used later in servicing the command.

If the command is a read and it can be serviced entirely from cache (i.e. a cache hit), the command is serviced by the read-hit portion of the controller (see description of read-hit handling).

If any portion of the read cannot be serviced from cached tracks (i.e. a cache miss), the command is turned over to the read- miss portion of the controller (see description of the read-miss handling).

If the command is a write and all tracks involved in the operation are already in cache, the command is serviced by the write-hit portion of the controller (see description of write-hit handling).

If any portion of the write involves an uncached track or tracks, the command is turned over to the write-miss portion of the controller (see description of the write-miss handling).

If the command is a seek, and the target track is already cached, no action is required. If the target track is not cached, the command is turned over to the seek-miss portion of the controller (see description of seek-miss handling).

Analyze Host I/O Command

As depicted in Figure 6, the analysis of a host command includes creation of a track address list which contains the locations of each track involved in the operation (see description of track address list setup). For each such track, the list contains the track's current location in cache, if it already resides there; or where it will reside in cache after this command and related caching activity have been completed. In the case that a track is not already cached, the space for it to be put into in cache is located, and the current track resident in that space is decached. The analysis includes setting the cache hit/miss flag so that the controller logic can be expedited.

Set Up Track Address List

As shown in Figure 7, the controller segment which sets up the track address list uses the I/O sector address and size to determine the disk track identifying numbers for each track involved in the I/O operation (see description of address translation). The number of tracks involved is also determined, and for each, the

portion of the track which is involved in the operation is calculated.

Address Translation

Figure 8 describes the operation for this translation. A sector address can be converted into a track address by dividing it by the track size. The quotient will be the track number, and the remainder will be the offset into the track where the sector resides.

Read-Hit Operation

Refer to Figure 9. A read hit is satisfied entirely from the cached data. In order to reach this module of the controller, the command will have been analyzed and the track address table will have been set up. With this preliminary work completed, the host read command can be satisfied by using each line of the track address table as a subcommand control. Since all required portions of all affected tracks are already in cache, all required data can be sent directly from the cache to the host. In addition to transferring the data to the host, this module will rechain the affected tracks to become the most-recently-used tracks in the LRU table.

Read-Miss Operation

A read miss (Fig. 10) is satisfied in part or wholly from the disk. In order to reach this module of the controller, the command will have been analyzed and the track address table will have been set up. With this preliminary work completed, the host read command can be satisfied by using each line of the track address table as a subcommand control. For an I/O whose size exceeds the READ-MISS-MAXSIZE, uncached portions are sent directly from the disk to the host without affecting the cache in any way. For an I/O whose size does not exceed the READ-MISS-MAXSIZE, the operation is handled based on the unit's current mode.

If the unit is not in urgent mode: For track segments which are already in cache, the data can be sent directly from the cache to the host. For a track segment not resident in the cache, the data is sent from the disk to the host, and simultaneously, the portion of the track from the first sector of the requested data to the end of that track is sent to the cache. The LRU-CACHED-LOW and LRU-CACHED-HIGH fields of the corresponding LRU table line(s) are set to reflect the portions of those tracks which have been brought into cache.

If the unit is in urgent mode: For a track not resident in the cache, the data is sent directly from the disk to the host without being entered into the cache.

In either mode, in addition to transferring the data to the host, this module will rechain affected, cached tracks to become the most-recently-used slots in the LRU table.

Write-Hit Operation

A write hit (Fig. 11) is handled entirely within the cache. In order to reach this module of the controller, the command will have been analyzed and the

track address table will have been set up. With this preliminary work completed, the host write command can be satisfied by using each line of the track address table as a subcommand control. Since all affected tracks are already represented in cache, all data can be sent directly from the host to the cache without any concern for post-transfer staging of partial tracks. In addition to transferring the data to the cache, this module will, if the slot was linked into the LRU chain, remove the affected cache slot from the LRU chain. In every case, the corresponding LRU-MOD-LOW, LRU-MOD-HIGH, LRU-CACHED-LOW, and LRU-CACHE-HIGH fields are updated to reflect the existence of this new data.

Write Miss Operation

If the I/O size exceeds the WRITE-MISS-MAXSIZE, uncached tracks or track segments are written directly to disk with no effect on the cache. For an I/O whose size does not exceed WRITE-MISS-MAXSIZE, the operation is handled based on the unit's current mode. If the unit is operating in normal mode, a write miss is handled entirely within the cache but requires the placing of information into the LRU-CACHED-LOW, LRU-CACHED-HIGH, LRU-MOD-LOW, and LRU-MOD-HIGH fields to allow for writing the data to disk, and to control the post-transfer staging of data from the disk into cache of any partial tracks that were involved, as depicted in Figure 12. In order to reach this module of the controller, the command will have been analyzed and the track address table will have been set up. With this preliminary work completed, the host write command can be satisfied by using each line of the track address table as a subcommand control. Since this is a cache-miss, some or all of the affected tracks are not in cache; however, all data can be sent directly from the host to the cache.

The cache controller has the responsibility for post-transfer staging of data to fill any gaps in

tracks, or any partial tracks. In actuality, only the first and/or last tracks involved in the transfer can be partial tracks; all interior tracks must be full tracks, and thus require no post-transfer staging in any case. For those tracks requiring post-transfer staging, the controller sets up a list of caching events to bring any required track segments into cache to maintain the integrity of the cached tracks. In addition to transferring the data to the cache, this module removes the affected tracks from the LRU chain. If the unit is operating in urgent mode, the handling of a write miss bypasses the cache for any tracks which are not currently cached, sending the data directly from the host to the disk. The LRU and ADT tables are updated for any cached tracks which may have been affected.

Seek Cache Miss

As shown in Figure 13, the controller has the option of ignoring a seek command since the controller will ultimately be responsible for fulfilling any subsequent, related I/O command. For a seek command for which the target track is already in cache, no controller action is needed or appropriate. For a seek command for which the target track is not in cache, the controller, if the disk to which the seek is directed is not busy, will cache that target track. This action is based on the assumption that the host would not send a seek command unless it was to be followed by a read or a write command. If the disk to which the seek is directed is busy when a seek command is received from the host, the seek command is ignored.

Decache a Track

For every cache-miss I/O that occurs, and for every cache-ahead operation, some previously cached track or tracks of data must be decached. The primary function of the LRU table is to allow the controller to expeditiously determine which cached track of data is

the best candidate for decaching. The decaching module depicted in Figure 14 chooses the track to be decached. Normally, the track with the longest elapsed time since its last usage will be the track to be decached. This is the track which is pointed to by the LRU-LRU element of the LRU table. The LRU-LRU pointer is maintained in such a way as to always point to the least-recently-used track.

The first condition is that the track must be inactive; that is, it is not at this instant the subject of any activity. It is highly unlikely that the LRU-LRU track would have any activity since most activities would reposition it out of the LRU-LRU spot. However, the possibility is covered by the decaching algorithm.

The second condition that must be satisfied before a track can be decached is that the track to be decached must not be a candidate for recycling. Any track which has been reused since it was first cached or reused since it was last recycled is not decached. Instead, such a cached track is recycled to the MRU position and its recycle marker is removed. In this manner, such a track is allowed to remain in cache for a longer time than a track which has not been reused since it was last recycled, or, in the case of a cache-ahead track, since the time it was first cached. The effect of this procedure is to allow unused cache-ahead tracks to move down through the LRU list at a faster rate than those which have been used, and to allow the more useful tracks to remain in cache for a longer time.

If, for any of the above reasons, the LRU-LRU track is unable to be decached, the LRU chain will point to the next-LRU candidate. While it is not unusual for the LRU track to be recycled, it will be an extremely unusual condition in which the decaching module will need to inspect more than one non-recycled LRU slot to find the slot to be decached.

When the actual candidate for decaching has been identified, both the LRU and ADT tables are updated to

reflect that the chosen candidate is no longer cached. This is a minor amount of work; no disk activity is involved.

Cache-Ahead Management

The cache hit and management operation is depicted in Figure 15. The controller attempts to cache-ahead after every host I/O which is a read operation regardless of whether the I/O was a cache hit or a cache miss. Operations which write data from the host to the device need no cache-ahead operations since data can always be accepted from the host into the cache's SSD. However, a read cache-ahead action is a background type of activity, and only uses the private channel between disk and cache, it will have a very minimal negative impact on the unit's response time to host I/O activity. To further limit the impact, the cache-ahead is given a lower priority than any incoming host I/O request.

A major factor in limiting the cache-ahead activity is the lack of need for its operation following most host I/O's. As depicted in the flow chart of Figure 16, the I/O locality feature limits the cache-ahead activity to those circumstances where it is likely that a near future host I/O would access data beyond the track which satisfied the current I/O. The I/O locality algorithm uses the current I/O block size and its location within the affected track to determine how near the "edge" of the current track the current I/O was located. If it is determined that a sufficient number of like-sized I/O's could be satisfied from this same track without need for an adjacent track, no cache-ahead is performed.

There are only two candidates for cache-ahead: they are the single track immediately following that involved in the host I/O and the track immediately preceding that of the host I/O. Since these tracks will often have already been cached by previous cache-ahead activity,

the cache-ahead activity is largely a self-limiting process.

Only one track is cached-ahead for any given host I/O: the track succeeding the host I/O is the primary candidate. If it is not already cached, and the proximity factor indicates the cache-ahead should occur, the forward track is cached at this time. If the succeeding track is already cached, the track preceding the host I/O is considered; if it is not already cached, and the proximity factor favors caching, this preceding track is cached at this time. Of course, if both of these candidate tracks had been cached previously, the cache-ahead module has no need to do any caching.

A very important benefit accrues from this cache-ahead, cache-back feature. If related tracks are going to be accessed by the host in a sequential mode, that sequence will be either in a forward or backward direction from the first one accessed in a given disk area. By the nature of the cache-ahead algorithm, an unproductive cache-ahead will only involve one track which lies in the wrong direction from the initial track in any given track cluster. This, coupled with the proximity algorithm, makes the cache-ahead behavior self-adapting to the direction of the accesses.

Background Sweep Management

When a write I/O from the host is serviced by the controller, the data from the host is placed in the cache. It is written from the cache to the disk in the background, minimizing the impact of the disk operations on the time required to service the I/O. The module that handles this background activity is the background sweep module. To limit the sweep activity, and thus limit contention for the spindle, only those portions of tracks which have been modified are written from SSD to disk during a sweep. In the interest of further efficiency, the background sweep module does not always copy data from cache to disk as soon as it is available.

Rather, it remains dormant until some minimum number of modified tracks are waiting to be copied before going into action. In order to avoid having a single modified track wait an inordinately long time before being copied from cache to disk, the background sweep will also be activated by a timeout. Thus, if any modified track has been waiting a certain minimum time, and the sweep is not active, the sweep will be activated. After the sweep has copied all modified portions of tracks from cache to disk, it returns to a dormant state.

Sweep Timeout

A timeout occurs when some cached data track has been modified and the corresponding track on disk has not been updated after a certain minimum time has elapsed. When a timeout occurs, by definition there will be at least one cached track which needs to be copied to disk. At this time, the background will be changed into the active state. The timeout module (Fig. 17) also causes the first background event to be set up (see description of background event generation), and if no conflict exists with the host for access to the disk, the execution of the event will be initiated. After this event is initiated, the next event, if one is known to be needed, is also set up and held for later execution. When these things have been done, the background sweep waits for circumstances to cause it to continue its operation or to return to a dormant state.

Sweep Initiation

At the completion of each host I/O operation, the sweep initiation module (Fig. 18) is entered. One of three cases may exist. The first case is that the sweep is dormant, and there are not a sufficient number of modified tracks waiting to be copied to disk to cause the sweep to be enabled at this time. In this case, which is the most common one, there is no action to be taken at this time.

In the second case, the sweep is active, and a background event is operating. In this situation, no new action is needed at this time.

In the final case, the sweep is active, but no background event is currently in operation. Under these conditions, a background event is generated (see description of Generate Sweep Event) and, if appropriate, its execution is initiated.

Generate Sweep Event

The need for the generation of a background sweep event is predicated on there being no other ongoing activity involving the disk. If the event generation module of Fig. 19 is entered when any such activity is in progress, no event is generated.

At times, the event generation module will find that there are no more modified tracks waiting to be copied to the disk. In this case, the background sweep is returned to the dormant condition. At other times, the background sweep is in the active mode, but has been temporarily interrupted to handle the higher priority activity of servicing a host I/O. Such interruption requires the background sweep to be restarted. It does this by finding the modified track which is nearest, but not directly at, the disk head; initiating a seek to that track; and then setting up a write event for the track. This write event will not be initiated until later, but its existence signals the sweep continuation module (see description of continuation module) that, if possible, this write is the next thing to be done.

The effect of this method of handling background writes is to minimize the impact on the host operations. The controller has an opportunity to service host I/O misses between the background seek and the corresponding write operation. None of this has any significant effect on servicing host I/O cache hits since hits are always handled immediately. The disk is not involved in a hit.

The sweep handles the writing of modified tracks differently depending on whether all the sectors in the track have been modified, or only some of the sectors have been modified. Further, the number of wholly modified tracks and the number of partially modified tracks are both taken into consideration in the setting of priorities for writing individual tracks to disk. When a larger number of wholly modified tracks exist, as opposed to the number partially modified, the wholly modified tracks are given preference by the sweep operation.

Writing a modified track from cache to disk is limited to handling only the modified portion of the track as defined by the corresponding LRU-MOD-LOW and LRU-MOD-HIGH values. Once the modified track or track segment has been written to disk, the track's cache slot, which has been in an unchained status, is placed in the LRU chain at the MRU position if the track had been only partially modified, and is placed at the LRU position if the track had been wholly modified. At the same time, the corresponding LRU-MOD-LOW and LRU-MOD-HIGH fields are set to their null value to indicate that no part of the cached data differs from that in the corresponding disk track.

Sweep Continuation

As depicted in the flow chart of Figure 20, each background sweep event, whether a seek or a write, prepares a waiting event for the sweep's subsequent action. Thus, the initiation of a seek always prepares the subsequent, related write event; the initiation of a write prepares the subsequent, unrelated seek event, if another track is waiting to be copied to disk.

The continuation module is entered upon the completion of each sweep event. If the host has issued an I/O command which requires the disk (in other words, a cache-miss), the background sweep sequence is interrupted, and the waiting event is erased. This action is

taken in order to expedite the servicing of the host's commands, and is taken regardless of the type of sweep event which is waiting. It can result in wasting background seek actions. This is acceptable; the aborted write will be handled later when time permits. Of course, once a sweep command, whether a seek or a write, has actually been initiated, it cannot be aborted.

If the sweep continuation module is entered after the sweep operations have been interrupted, it will use the event generation module (see description of event generation) to restart the sweep sequence.

Finally, if the continuation module finds that the just completed sweep operation was a write, and no more modified tracks are waiting to be copied to the disk, the sweep is put into the dormant state.

Power Down Control

As depicted in the flow chart of Figure 21, this portion of the firmware is invoked when the unit senses that the line power to it has dropped. Since some of the data in the unit may be in the cache portion in a modified state and awaiting transfer to the disk, power must be maintained on the cache memory until the modified portions have been written to the disk. Thus, a failure of the line power causes the unit to switch to the battery backup unit. The battery backup unit provides power while the memory device goes through an intelligent shutdown process.

If the host is in the process of a data transfer with the memory device when power drops, the shutdown controller allows the transfer in progress to be completed. It then blocks any further transactions with the host from being initiated.

The shutdown controller then must initiate a background sweep to copy any modified portions of data tracks from the solid state memory to the disk so that it will not be lost when power is completely shut off to

the control and memory circuits. After the sweep is completed (which will take only a few seconds), all data in the solid state memory will also reside on the disk. At this point the disk spindle can be powered down, reducing the load on the battery.

Most power outages are of a short duration. Therefore, the controller continues to supply battery power to the control circuits and the solid state memory for some number of seconds. If the outside power is restored in this time period, the controller will power the spindle back up and switch back to outside power. In this case, the operation can proceed without having to reestablish the historical data in the solid state memory. In any case, no data is at risk since it is all stored on the rotating magnetic disk before final shutdown.

Final Background Sweep

The final background sweep (Fig. 22) copies modified portions of tracks from the solid state memory to the magnetic disk. There will usually be only a few such tracks, or portions of tracks to copy since the number that can reach this state is intentionally limited by the operations of the system. The final sweep makes use of logic developed for the normal operation of the background sweep.

The sweep is initiated in much the same manner as for a timeout during normal operation. If no tracks need to be copied, the sweep is left in the dormant state, and no further sweep action is required. If any tracks need copied, the sweep initiator sets up and initiates the first background seek, as well as sets up the related write event. At the completion of this first seek, control goes to the background continuation module which alternately executes the previously created, waiting event and generates the next event and puts it into a wait status. When no modified tracks remain to be copied, the sweep is finished.

Parameters and Particulars

This specification refers to items which are not given specific quantities or identities. These have been purposely left unquantified so as not to imply any absolute limits or restrictions. For purposes of illustration, and to provide known workable dimensions and identities, the following ranges of values and identifiers are provided, along with a set which is satisfactory for a sample configuration.

BACKGROUND SWEEP TRIGGER, NUMBER OF MODIFIED TRACKS
Range: One to number of tracks on chosen disk.
Sample configuration: Five

BACKGROUND SWEEP TRIGGER, TIME
Range: One millisecond to unlimited.
Sample configuration: Five seconds.

EPROM MEMORY FOR MICROPROCESSOR
Size range: Non-specific.
Sample configuration: 64 kilobytes.

HARDWARE MICROPROCESSOR CONTROLLER
Candidates: Any suitable and available microprocessor.
Sample configuration: 80C196, 24 MHz.

POWER DOWN, CACHE HOLD TIME
Range: Zero seconds to limitation imposed by battery backup unit.

Sample configuration: Five minutes.

ROTATING MAGNETIC DISK CAPACITY
Size range: Any available disk capacity.

Sample configuration: 675 megabytes formatted.

SCSI CONTROLLER
Candidates: Any suitable and available controller device. Sample configuration: NCR 53C90A.

SECTOR SIZE
Size range: Any appropriate for the host system and the selected disk drive.
Sample configuration: 180 bytes.

SECTORS PER TRACK
Range: Any appropriate for selected disk and host system.
Sample configuration: 256.

SOLID STATE MEMORY SIZE
Size range: One megabyte to 100 percent of the capacity of the attached disk capacity.
Sample configuration: 32 megabytes.

TRACK SIZE
Size range: One sector to any size appropriate for the selected disk drive.
Sample configuration: 256 sectors.

TRACKS PER DISK
Range: Any available on chosen disk.
Sample configuration: 14650.

TABLE FORMATS

TABLE F-1

ADDRESS TRANSLATION (ADT) TABLE FORMAT - UNINDEXED

ELEMENTS

TABLE ITEM	DESCRIPTION
ADT-CNL	Number of tracks on the cached disk spindle; equals the number of lines in the ADT table.
ADT-HEAD-POS	Position of read/write head of cache disk.
ADT-SWEEP-DIR	Direction of current DISK SERVER sweep; 1 = sweep is progressing from low-to-high. 0 = sweep is progressing from high-to-low.
ADT-MOD-COUNT	Total number of tracks in the cache which have been modified by writes from the host and are currently awaiting a write to disk by the Disk server.
ADT-MOD-URGENT	The number of cache slots which, when in a modified condition, causes the device to shift priorities to maintain optimal performance.
ADT-READ-HITS	Number of cache read-hits encountered since last reset.
ADT-READ-MISSES	Number of read-misses encountered since last reset.
ADT-WRITE-HITS	Number of write-hits encountered since last reset.
ADT-WRITE-MISSES	Number of write-misses encountered since last reset.

TABLE F-2

ADDRESS TRANSLATION TABLE FORMAT - INDEXED ELEMENTS

TABLE ITEM	MAXIMUM VALUE	ITEM DESCRIPTION
(INDEX)	(ADT-CNL)	ADT table index; equivalent to the corresponding disk track number. There is one ADT table line for each disk track.
ADT-SLOT	(LRU-CNL)	Number of the cache slot which contains the disk track of data corresponding to this ADT index; also points to line in LRU table related to the disk track. If the disk track is not in cache, this field is set to its null value to indicate that fact.
ADT-MODIFIED	1	Flag indicating whether or not this (cached) track has been modified by a write operation from the host, and thus, needs to be written from the cache to the disk. 0 = This track (if cached) is unmodified and does not need written to disk. 1 = This track needs written to disk.

TABLE F-3

LEAST-RECENTLY-USED (LRU) TABLE FORMAT - UNINDEXED ELEMENTS

TABLE ITEM	DESCRIPTION
LRU-CNL	Number of lines in the LRU table; equal to the number of slots in the cache area.
LRU-LRU	Pointer to least-recently-used end of the LRU chain.
LRU-MRU	Pointer to most-recently-used end of the LRU chain.

TABLE F-4

LEAST-RECENTLY-USED TABLE FORMAT - INDEXED ELEMENTS

TABLE ITEM	MAXIMUM VALUE	ITEM DESCRIPTION
LRU-TRACK	(ADT-CNL)	Disk track number for data stored in this cache slot; also points to line in ADT table related to the disk track.
LRU-NEXT	(LRU-CNL)	Pointer to following link in LRU chain; 0 = this is last (MRU) link in chain.
LRU-TRACK	(ADT-CNL)	Disk track number for data stored in this cache slot; also points to line in ADT table related to the disk track.
LRU-CACHED-LOW	(TRACK SIZE)	Lowest track-relative sector number within the cached track which contains valid data.
LRU-CACHED-HIGH	(TRACK SIZE)	Highest track-relative sector number within the cached track which contains valid data.
LRU-MOD-LOW	(TRACK SIZE)	Lowest track-relative sector number within the cached track which contains modified data.
LRU-MOD-HIGH	(TRACK SIZE)	Highest track-relative sector number within the cached track which contains modified data.
LRU-LOCKED	1	Flag indicating whether or not this (cached) track is

			currently the target of some operation, such as being acquired from the disk, being written to the disk by the cache controller. 0 = the (cached) track is not locked; it is available for any operations. 1 = the (cached) track is locked; it cannot, at this moment, become the target of another, conflicting operation.
	LRU-RECYCLE	1	Recycle marker; 0 = this is not a candidate for recycling

TABLE T-1

INITIAL ADT TABLE

The ADT TABLE is set to initial conditions to indicate that no disk tracks are cached. Note: A "*" indicates a null value.

ADT-CNL	= 14628
ADT-HEAD-POS	= 0
ADT-SWEEP-DIR	= 1
ADT-MOD-COUNT	= 0
ADT-MOD-URGENT	= 16
ADT-READ-HITS	= 0
ADT-READ-MISSES	= 0
ADT-WRITE-HITS	= 0
ADT-WRITE-MISSES	= 0

DISK TRACK	SSD SLOT	MODIFIED
1	*	0
2	*	0
3	*	0
4	*	0
5	*	0
6	*	0
.	.	.
.	.	.
.	.	.

(ADT-CNL)

TABLE T-2

INITIAL LRU TABLE

The LRu TABLE is arbitrarily chained to allow initial operations to proceed with a minimum of special handling during startup of the caching operations. Table is listed in MRU-to-LRU order. Note: A "*" indicates a null value.

CNL = 22
LRU = 1
MRU = 22

SSD SLOT	LRU LAST	LRU NEXT	DISK TRACK	-CACHED- LOW	-CACHED- HIGH	MODIFIED LOW	MODIFIED HIGH	LRU LOCK	RE-CYCLE
22	21	0	0	*	*	*	*	0	0
(Slot 22 is arbitrarily designated the MRU)									
21	20	22	0	*	*	*	*	0	0
20	19	21	0	*	*	*	*	0	0
19	19	20	0	*	*	*	*	0	0
18	17	19	0	*	*	*	*	0	0
17	16	18	0	*	*	*	*	0	0
16	15	17	0	*	*	*	*	0	0
15	14	16	0	*	*	*	*	0	0
14	13	15	0	*	*	*	*	0	0
13	12	14	0	*	*	*	*	0	0
12	11	13	0	*	*	*	*	0	0
11	10	12	0	*	*	*	*	0	0
10	9	11	0	*	*	*	*	0	0
9	8	10	0	*	*	*	*	0	0
8	7	9	0	*	*	*	*	0	0
7	6	8	0	*	*	*	*	0	0
6	5	7	0	*	*	*	*	0	0
5	4	6	0	*	*	*	*	0	0
4	3	5	0	*	*	*	*	0	0
3	2	4	0	*	*	*	*	0	0
2	1	3	0	*	*	*	*	0	0
1	0	2	0	*	*	*	*	0	0
(Slot 1 is arbitrarily designated the LRU)									

TABLE T-3a

ADT TABLE AFTER ONE I/O OPERATION
(A read involving tracks 46 and 47)
Note: A "*" indicates a null value.

ADT-CNL	= 14628
ADT-HEAD-POS	= 47
ADT-SWEEP-DIR	= 1
ADT-MOD-COUNT	= 0
ADT-MOD-URGENT	= 16
ADT-READ-HITS	= 0
ADT-READ-MISSES	= 1
ADT-WRITE-HITS	= 0
ADT-WRITE-MISSES	= 0

DISK TRACK	SSD SLOT	MODIFIED	COMMENTS
1	*	0	
2	*	0	
3	*	0	
4	*	0	
5	*	0	
6	*	0	
.	.	.	
.	.	.	
.	.	.	
46	1	0	from read-miss (2-track)
47	2	0	from read-miss (2-track)
48	*	0	
49	.	.	
(ADT-CNL)	.	.	

TABLE T-3b

LRU TABLE AFTER ONE READ I/O OPERAT^FON
(A read involving track 46)

LRU-CNL = 22
LRU-LRU = 3
LRU-MRU = 2

SSD SLOT	LRU LAST	LRU NEXT	DISK TRACK	-CACHED- LOW	HIGH	MODIFIED LOW	HIGH	LRU LOCK	RE-CYCLE
2	1	0	47	1	256	*	*	0	0
(Slot 2 becomes the new MRU)									
1	22	2	46	222	256	*	*	0	0
(Slots 1 and 2 have been used to cache the 2-track read-miss.)									
22	21	1	*	*	*	*	*	0	0
(Slot 22 was old MRU)									
21	20	22	*	*	*	*	*	0	0
20	19	21	*	*	*	*	*	0	0
19	18	20	*	*	*	*	*	0	0
18	17	19	*	*	*	*	*	0	0
17	16	18	*	*	*	*	*	0	0
16	15	17	*	*	*	*	*	0	0
15	14	16	*	*	*	*	*	0	0
14	13	15	*	*	*	*	*	0	0
13	12	14	*	*	*	*	*	0	0
12	11	13	*	*	*	*	*	0	0
11	10	12	*	*	*	*	*	0	0
10	9	11	*	*	*	*	*	0	0
9	8	10	*	*	*	*	*	0	0
8	7	9	*	*	*	*	*	0	0
7	6	8	*	*	*	*	*	0	0
6	5	7	*	*	*	*	*	0	0
5	4	6	*	*	*	*	*	0	0
4	3	5	*	*	*	*	*	0	0
3	0	4	*	*	*	*	*	0	0
(Slot 3 becomes the new LRU)									

TABLE T-3c

LRU TABLE AFTER 1000 I/O OPERATIONS
I/O 1000 was a read involving track 56.
Table is listed in MRU-to-LRU order.
Note: A "*" indicates a null value.

LRU-CNL = 22
LRU-LRU = 3
LRU-MRU = 21

SSD SLOT	LRU LAST	LRU NEXT	DISK TRACK	-CACHED- LOW	HIGH	MODIFIED LOW	HIGH	LRU LOCK	RE-CYCLE
21	18	0	56	1	256	*	*	0	1
(read-hit on a cache-ahead slot)									
18	19	21	213	1	256	*	*	0	0
(cleaned by writing and modified portion to disk)									
19	5	18	212	227	256	*	*	0	0
5	17	19	8071	255	256	*	*	0	0
17	1	5	63	1	256	*	*	0	0
(cached-ahead for read of track 62)									
1	8	17	62	1	256	*	*	0	0
(cached by read miss spanning tracks 61-62)									
8	9	1	61	191	256	*	*	0	0
9	14	8	48	117	256	*	*	0	0
14	20	9	65	1	256	*	*	0	0
(cached-backward for read of track 66)									
20	16	14	66	135	256	*	*	0	0
(cached due to read-miss of track 66)									
16	2	20	57	127	256	*	*	0	0
2	12	16	46	153	256	*	*	0	0
12	22	2	52	181	256	*	*	0	0
22	4	12	67	1	256	*	*	0	0
4	15	22	41	1	256	*	*	0	0
15	10	4	42	21	256	*	*	0	0
10	6	15	43	1	256	*	*	0	0
6	3	10	58	1	256	*	*	0	0
3	0	6	215	1	256	*	*	0	0
(slot 3 is now the LRU slot)									

Following slots have been modified but not yet cleaned by writing modified portion to disk; thus, they are not chained.

7	*	*	45	1	256	1	2	0	0
11	*	*	44	191	256	191	256	0	0
13	*	*	214	55	256	55	122	0	0

TABLE T-3d

LRU TABLE AFTER 1000 I/O OPERATIONS
I/O 1000 was a read involving track 56.
Table is listed in MRU-to-LRU order.
Note: A "*" indicates a null value.

LRU-CNL = 22
LRU-LRU = 3
LRU-MRU = 21

SSD SLOT	LRU LAST	LRU NEXT	DISK TRACK	-CACHED- LOW	HIGH	MODIFIED LOW	HIGH	LRU LOCK	RE-CYCLE
21	18	0	56	1	256	*	*	0	1
(slot 21 is still the MRU)									
18	19	21	213	1	256	*	*	0	0
19	5	18	212	227	256	*	*	0	0
5	17	19	8071	255	256	*	*	0	0
17	9	5	63	1	256	*	*	0	0
9	14	17	48	117	256	*	*	0	0
14	20	9	65	1	256	*	*	0	0
20	16	14	66	135	256	*	*	0	0
16	2	20	57	127	256	*	*	0	0
2	12	16	46	153	256	*	*	0	0
12	22	2	52	181	256	*	*	0	0
22	4	12	67	1	256	*	*	0	0
4	15	22	41	1	256	*	*	0	0
15	10	4	42	21	256	*	*	0	0
10	6	15	43	1	256	*	*	0	0
6	3	10	58	1	256	*	*	0	0
3	0	6	215	1	256	*	*	0	0
(slot 3 is still the LRU slot)									

Following slots have been modified but not yet cleaned by writing modified portion to disk; thus, they are not chained. Since five tracks have been modified, the background sweep will be turned on.

1	*	*	62	1	256	1	2	0	0
7	*	*	45	1	256	1	2	0	0
8	*	*	61	191	256	191	256	0	0
11	*	*	44	191	256	191	256	0	0
13	*	*	214	55	256	55	122	0	0

TABLE T-3e

LRU TABLE AFTER 1002nd I/O OPERATION
(A write involving track 214)
Table is listed in MRU-to-LRU order.
Note: A "*" indicates a null value.

LRU-CNL = 22
LRU-LRU = 3
LRU-MRU = 11

SSD SLOT	LRU LAST	LRU NEXT	DISK TRACK	-CACHED- LOW	HIGH	MODIFIED LOW	HIGH	LRU LOCK	RE-CYCLE
11	7	0	44	191	256	*	*	0	0
(slot 11 is new MRU, based on cleaning operations)									
7	8	11	45	1	256	*	*	0	0
8	1	7	61	191	256	*	*	0	0
1	21	8	62	1	256	*	*	0	0
21	18	1	56	1	256	*	*	0	0
18	19	21	213	1	256	*	*	0	0
19	5	18	212	227	256	*	*	0	0
5	17	19	8071	255	256	*	*	0	0
17	9	5	63	1	256	*	*	0	0
9	14	17	48	117	256	*	*	0	0
14	20	9	65	1	256	*	*	0	0
20	16	14	66	135	256	*	*	0	0
16	2	20	57	127	256	*	*	0	0
2	12	16	46	153	256	*	*	0	0
12	22	2	52	181	256	*	*	0	0
22	4	12	67	1	256	*	*	0	0
4	15	22	41	1	256	*	*	0	0
15	10	4	42	21	256	*	*	0	0
10	6	15	43	1	256	*	*	0	0
6	3	10	58	1	256	*	*	0	0
3	0	6	215	1	256	*	*	0	0
(slot 3 is still the LRU slot)									

Following slots have been modified but not yet cleaned by writing modified portion to disk; thus, they are not chained. Since only 1 track is modified, the background sweep will remain inactive.

13	*	*	214	55	256	55	122	0	0

TABLE 4

Sample I/O's for Illustration

The LRU and ADT table examples are based on I/O samples taken from an actual operating computer system and projected into the system's environment.

For each I/O, the following information is available:

(I/O REF NBR)	SECTOR ADDRESS	SIZE IN SECTORS	(COMPUTED TRACK NUMBER)	comment
1	11,742	68	46, 47	read starts in 46, ends in 47
.	.	.	.	
.	.	.	.	
.	.	.	.	
1000	14,190	68	56	read completely in 56
1001	15,550	68	61, 62	write starts in 61, ends in 62
1002	54,582	68	214	write entirely in 214
.	.	.	.	
.	.	.	.	
.	.	.	.	

All publications and patent applications mentioned in this specification are herein incorporated by reference to the same extent as if each individual publication or patent application was specifically and individually indicated to be incorporated by reference.

The invention now being fully described, it will be apparent to one of ordinary skill in the art that many changes and modifications can be made thereto without departing from the spirit or scope of the appended claims.

WHAT IS CLAIMED IS:

1. A cache memory system comprising:

a mass storage device;

a cache memory;

a power subsystem for providing power to said mass storage device and said cache memory, said power subsystem receiving power from a normal power source and an auxiliary power source;

means for detecting when said normal power source to said power subsystem is interrupted;

means for ceasing normal operation of said cache memory when said normal power source is interrupted;

means for writing data from said cache memory to said mass storage device when said interruption is detected; and

means for shutting down power from said power subsystem to said mass storage device and said cache memory after said data is written from said cache memory to said mass storage device after said interruption.

2. A method for operating a cache memory system having a mass storage device and a cache memory, said method comprising the steps of:

storing in said cache memory selected data from said mass storage device;

maintaining a Least Recently Used (LRU) table indicating the relative recency of use of each data stored in said cache memory;

adding an entry to the top of said LRU table for each new data stored in said cache memory;

setting a flag for each entry in said LRU table when data stored in said cache memory associated with said LRU entry has been used;

for each LRU table entry which reaches the bottom of said LRU table:

determining if said flag has been set;

if said flag has been set, unsetting said flag and placing said entry at the top of said LRU table; and

if said flag has not been set, deleting said entry from said LRU table and decaching said data in said cache memory associated with said LRU table entry.

3. A method as in claim 2 which further comprises the step of setting said flag for a selected LRU table entry when said LRU table entry is added to said LRU table.

4. A method for operating a cache memory system having a mass storage device and a cache memory, said method comprising the steps of:

seeking access to data stored within said mass storage device;

determining a first distance between a second side of a first selected subsection of said mass storage device containing the data for which access is sought and one side of the data for which access is sought;

determining the number of available I/O operations based upon said first distance and the amount of data is anticipated to be transferred in a single I/O operation;

determining if said number of available I/O operations is less than a preselected critical number; and

if said number of available I/O operations is less than said preselected critical number, copying into said cache memory data within a second selected subsection of said mass storage device located adjacent said first side of said first selected subsection.

5. A method as in claim 4 wherein said first distance is a forward distance measured from the end of the data for which access is sought and the start of an adjacent subsection of said mass storage device.

6. A method as in claim 4 wherein said first distance is a backward distance measured from the start of the data for which access is sought and the end of an adjacent subsection of said mass storage device.

7. A method as in claim 4 wherein said selected subsection of said mass storage device is a logical block.

8. A method as in claim 4 which further comprises the steps of:

if said number of available I/O operations is greater than said preselected critical number:

determining a second distance between a second side of said first selected subsection of said mass storage device containing the data for which access is sought and one side of the data for which access is sought;

determining the number of available I/O operations based upon said second distance and the amount of data which is anticipated to be transferred in a single I/O operation;

determining if said number of available I/O operations is less than a preselected critical number; and

if said number of available I/O operations is less than said preselected critical number, copying into said cache memory said data within a third selected subsection of said mass storage device located adjacent said second side of said first selected subsection.

9. A cache memory system comprising:

a mass storage device organized by logical blocks, each logical block including a plurality of sectors;

a cache memory for temporarily storing selected data associated with said mass storage device;

a logical block address look up table for storing information indicating which logical block information of said mass storage device corresponding to data currently stored in said cache memory, said logical block address look up table comprising a flag for each logical block of said mass storage device, said flag indicating whether data associated with said logical block is currently stored within said cache memory.

10. A cache memory system as in claim 9 wherein said logical block address look up table is indexed by a unique sequential logical block number associated with each logical block of said mass storage device.

11. A cache memory system as in claim 10 which further includes address translation means for converting logical block numbers of said mass storage device to said sequential logical block numbers of said logical block address look up table.

12. A cache memory system as in claim 11 wherein said address translation means comprises means for dividing a sequential sector number by the number of sectors per logical block of said mass storage device in order to provide a quotient equal to the logical block number and a remainder equal to the sector number within said logical block of said mass storage device corresponding to said sequential sector number.

13. A method for operating a cache memory system having a mass storage device and a cache memory, said method comprising the steps of:

organizing said mass storage device into a plurality of logical blocks, each logical block including a plurality of sectors; and

maintaining a logical block address look up table which includes one entry associated with each logical block of said mass storage device, each said

entry having an associated sequential logical block number and a flag indicating if data corresponding to an associated logical block of said mass storage device is currently stored in said cache memory.

14. A method as in claim 13 which further comprises the steps of:

accessing a desired sequential sector address within said logical block address look up table as a function of a selected sector within a selected logical block within said mass storage device by calculating the number of sectors from the start of said mass storage device to said selected sector within said selected logical block; and

using said calculation to derive said sequential sector address of said cache memory associated with said selected sector within said selected logical block of said mass storage device.

15. A method as in claim 13 which further comprises the steps of:

accessing a desired sector within a selected logical block within said mass storage device as a function of a selected sequential sector address within said logical block address look up table by dividing said selected sequential sector address by the number of sectors per logical block in said mass storage device to derive a quotient and a remainder;

using said quotient to derive said selected logical block of said mass storage device; and

using said remainder to derive said selected sector of said selected logical block of said mass storage device.

16. A method for operating a cache system having a mass storage device and a cache memory, said method comprising the steps of:

writing data to said cache in locations corresponding to selected locations in said mass storage device;

determining the elapsed time between writing data to said cache memory prior to being written to said mass storage device; and

if said elapsed time exceeds a predetermined period of time, writing data from said cache memory to said mass storage device.

17. A method for operating a cache system having a mass storage device and a cache memory, said method comprising the steps of:

writing data to said cache in locations corresponding to selected locations in said mass storage device;

determining the amount of data which has been written to said cache memory prior to being written to said mass storage device; and

if said amount of data exceeds a predetermined amount, writing data from said cache memory to said mass storage device.

18. A method as in claim 16 wherein said mass storage device comprises a plurality of tracks, and said step of writing data from said cache memory to said mass storage device comprises the step of writing data from said cache memory which has not yet been written to said mass storage device, said data being selected by its location relative to the currently selected track of said mass storage device.

19. A method as in claim 18 wherein said data is selected as that data corresponding to the closest track of said mass storage device with respect to said currently selected track.

20. A method as in claim 17 wherein said mass storage device comprises a plurality of tracks, and said step of writing data from said cache memory to said mass storage device comprises the step of writing data from said cache memory which has not yet been written to said mass storage device, said data being selected by its location relative to the currently selected track of said mass storage device.

21. A method as in claim 20 wherein said data is selected as that data corresponding to the closest track of said mass storage device with respect to said currently selected track.

22. A method for operating a cache memory system having a mass storage device and a cache memory, each organized as a plurality of blocks, each block having a plurality of sectors, said method comprising the steps of:

storing in said cache memory selected data from said mass storage device;

maintaining a Least Recently Used (LRU) table indicating the relative recency of use of each block of data stored in said cache memory and the sectors within said block which contains current data;

adding an entry to the top of said LRU table for each new block of data stored in said cache memory;

setting a flag for each block entry in said LRU table when data stored in said cache memory associated with said LRU entry has been used;

for each LRU block entry which reaches the bottom of said LRU table:

determining if said flag has been set;

if said flag has been set, unsetting said flag and placing said entry at the top of said LRU table; and

if said flag has not been set, deleting said entry from said LRU table and decaching said block of

data in said cache memory associated with said LRU table entry.

23. A method as in claim 22 which further comprises the step of maintaining data in said LRU table indicating which sectors of each block of data stored in said cache have been modified.

24. A method as in claim 22 wherein, for each block stored in said cache memory, sectors of cached data are maintained contiguous, with no sector gaps between cachced data, although not all sectors of a given block need be cached.

25. A cache memory system comprising:

a mass storage device organized by logical blocks, each logical block including a plurality of sectors;

a cache memory for temporarily storing selected data of a contiguous set of valid sectors associated with said mass storage device, said contiguous set of valid secors possibly including a contiguous set of modified sectors;

a logical block address look up table for storing information indicating which logical block information of said mass storage device corresponding to data currently stored in said cache memory, said logical block address look up table comprising a flag for each logical block of said mass storage device, said flag indicating whether data associated with said logical block is currently stored within said cache memory, said logical block look up table also storing data associated with each logical block indicating which sectors within said logical block contain valid sectors and modified sectors.

26. A cache memory system as in claim 25 wherein said logical block address look up table is indexed by a

unique sequential logical block number associated with each logical block of said mass storage device.

27. A cache memory system as in claim 26 which further includes address translation means for converting logical block numbers of said mass storage device to said sequential logical block numbers of said logical block address look up table.

28. A cache memory system as in claim 27 wherein said address translation means comprises means for dividing a sequential sector number by the number of sectors per logical block of said mass storage device in order to provide a quotient equal to the logical block number and a remainder equal to the sector number within said logical block of said mass storage device corresponding to said sequential sector number.

29. A cache memory system as in claim 22 wherein an LRU track entry is flagged as a dechained member of the LRU table if that track contains modified data.

30. A cache memory as in claim 29 wherein a track entry is made a chained member of the LRU table after its modified data is written to said mass storage device.

31. A cache memory as in claim 30 wherein said track entry is made a chained member at the bottom of the LRU table if every sector of said track was modified prior to writing to said mass storage device.

32. A cache memory as in claim 30 wherein said track entry is made a chained member at the top of the LRU table if less than every sector of said track was modified prior to writing to said mass storage device.

33. A method for operating a cache memory system having a mass storage device and a cahce memory, said method comprising the steps of:

storing in said cache memory selected data from said mass storage device;

reading data from said cache memory when desired data is available in said cache memory;

periodically updating the data stored in said cache memory based upon data desired to be accessed; and

accessing said mass storage device directly when data of a selected size is desired to be accessed.

34. A method as in claim 33 wherein said selected size is a function of the amount of unmodified data stored in said cache memory.

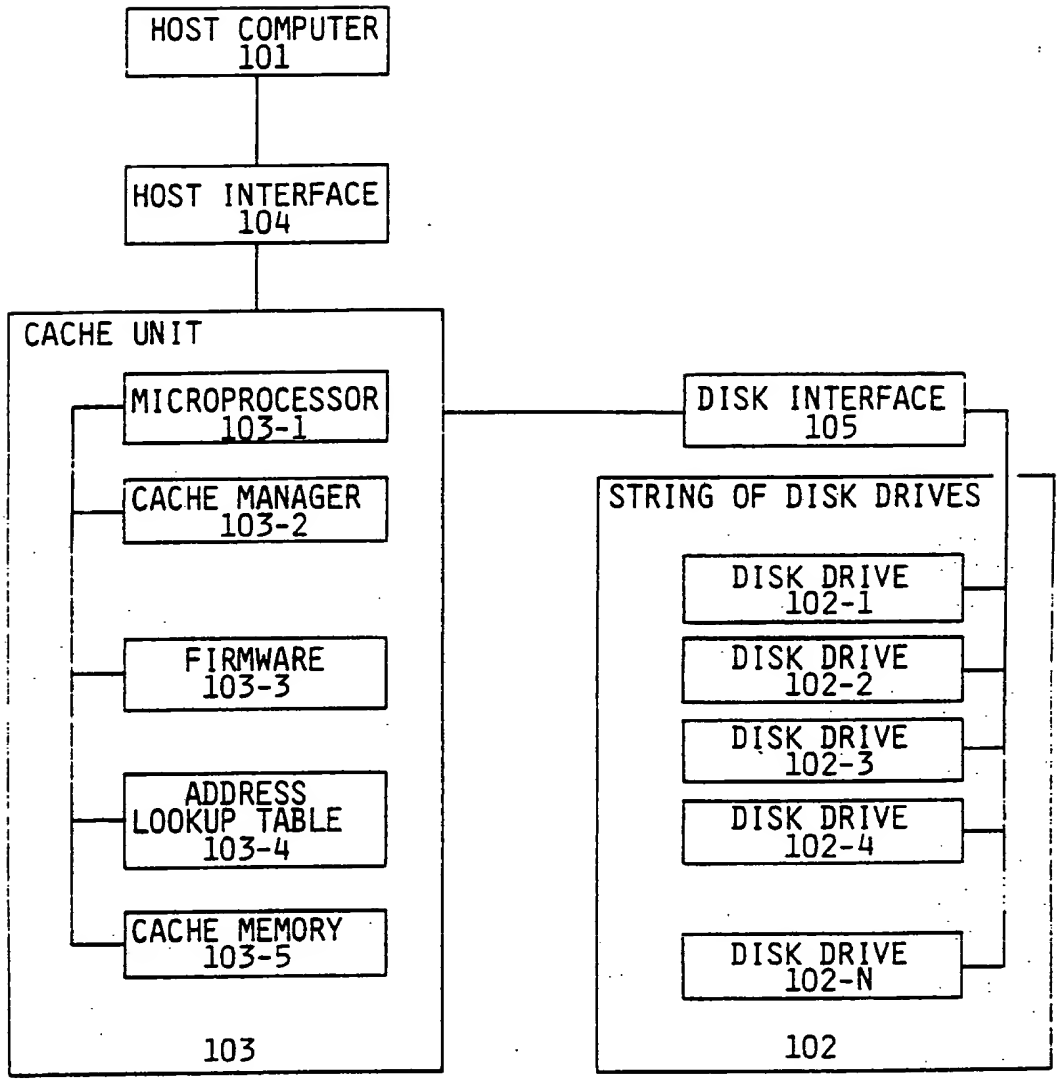


FIG. 1
PRIOR ART

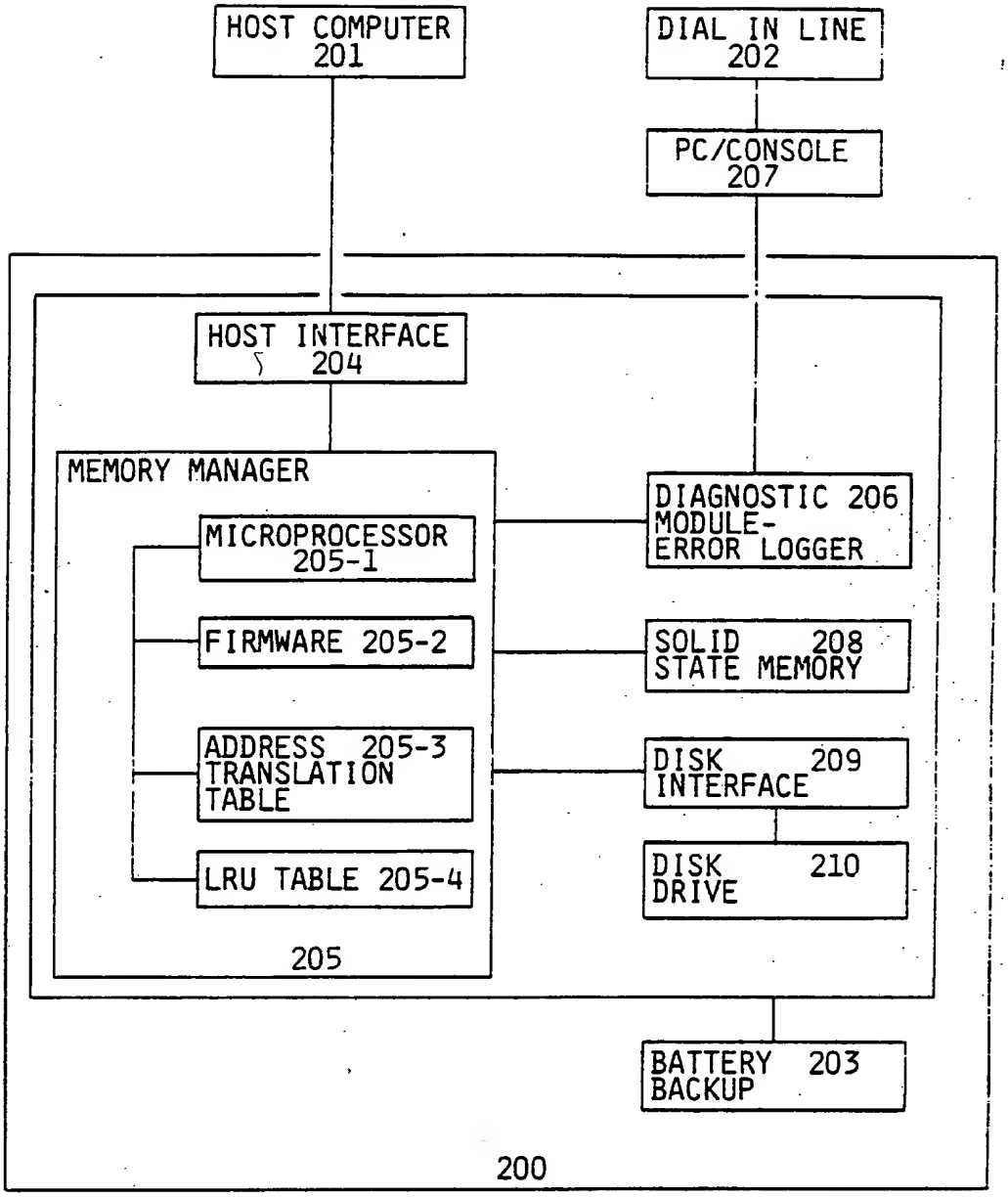

HOST COMPUTER 201
DIAL IN LINE 202
PC/CONSOLE 207
HOST INTERFACE 204
MEMORY MANAGER
MICROPROCESSOR 205-1
FIRMWARE 205-2
ADDRESS TRANSLATION TABLE 205-3
LRU TABLE 205-4
205
DIAGNOSTIC MODULE-ERROR LOGGER 206
SOLID STATE MEMORY 208
DISK INTERFACE 209
DISK DRIVE 210
BATTERY BACKUP 203
200

FIG. 2

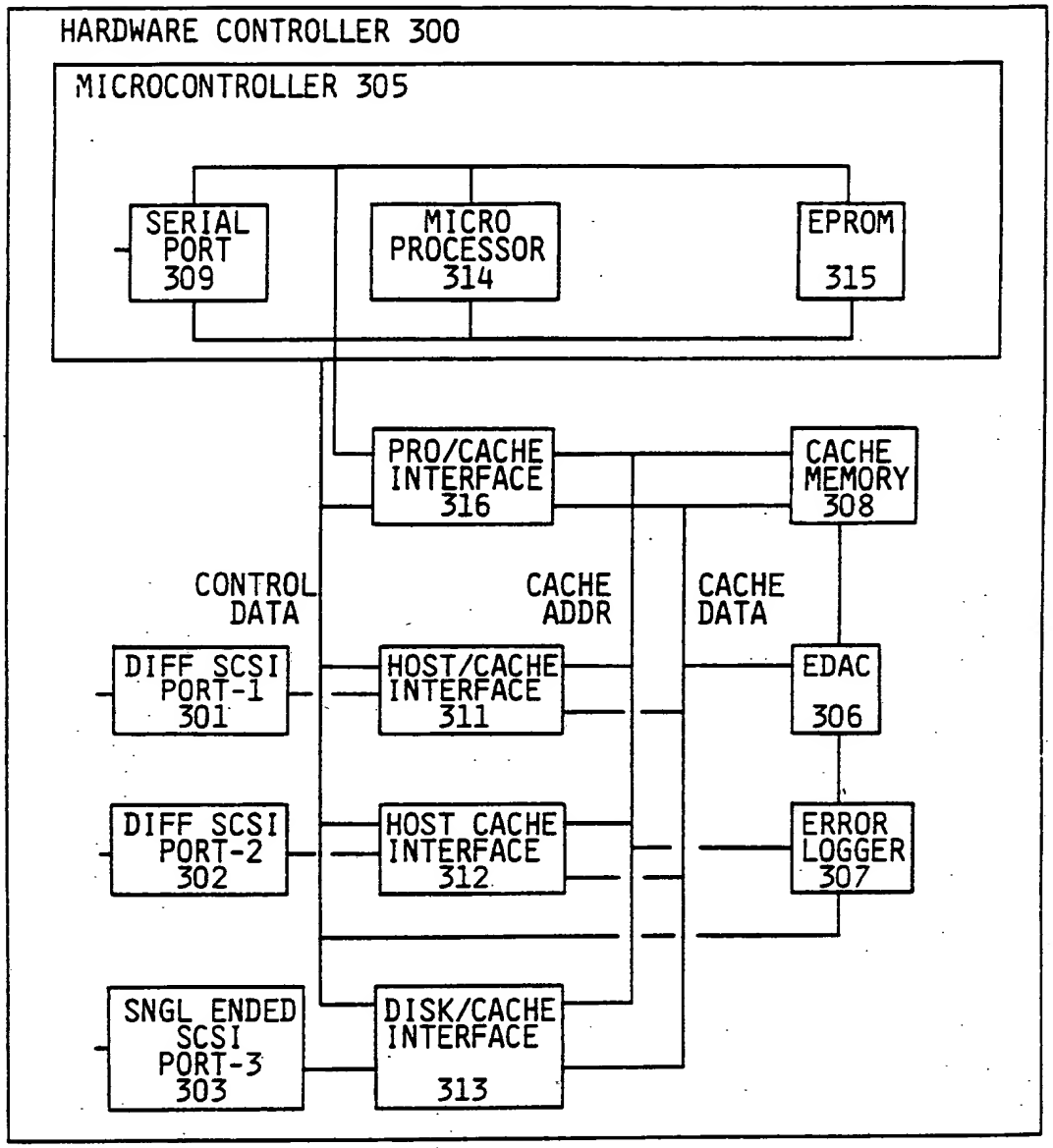

HARDWARE CONTROLLER 300
MICROCONTROLLER 305
SERIAL PORT 309
MICRO PROCESSOR 314
EPROM 315
PRO/CACHE INTERFACE 316
CACHE MEMORY 308
CONTROL DATA
CACHE ADDR
CACHE DATA
DIFF SCSI PORT-1 301
HOST/CACHE INTERFACE 311
EDAC 306
DIFF SCSI PORT-2 302
HOST CACHE INTERFACE 312
ERROR LOGGER 307
SNGL ENDED SCSI PORT-3 303
DISK/CACHE INTERFACE 313

FIG. 3

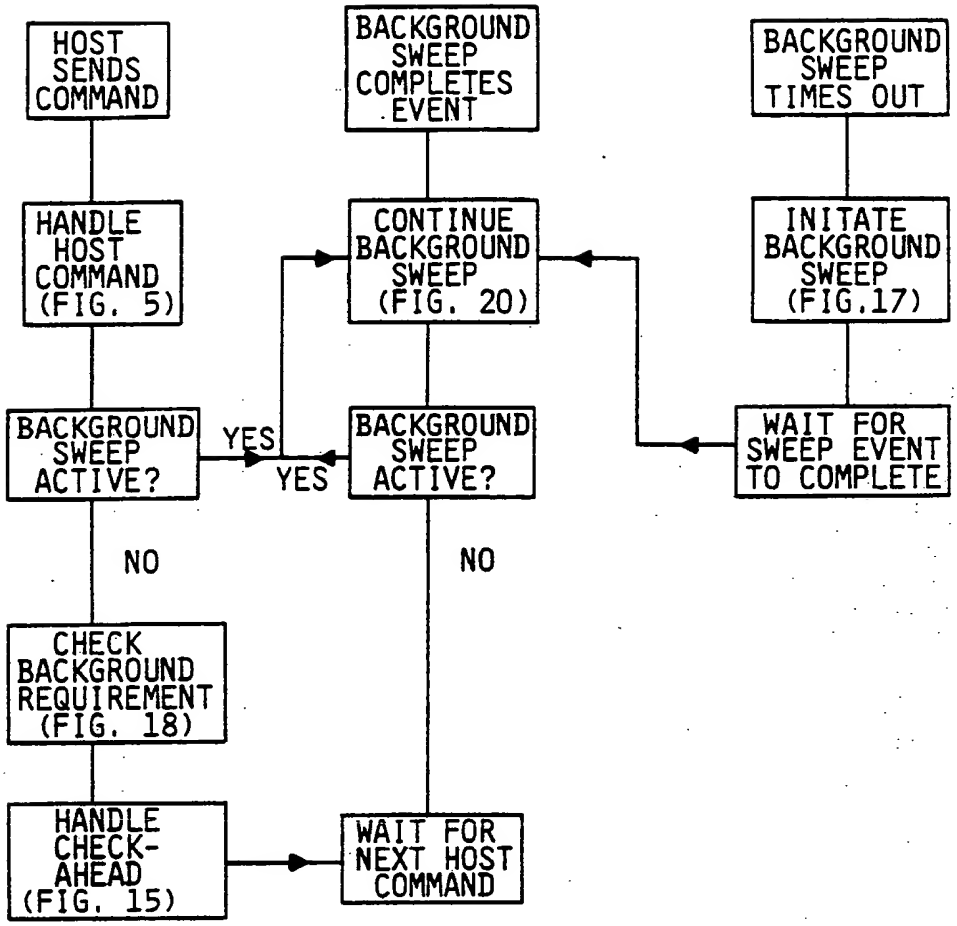

HOST SENDS COMMAND
HANDLE HOST COMMAND (FIG. 5)
BACKGROUND SWEEP ACTIVE?
YES
NO
CHECK BACKGROUND REQUIREMENT (FIG. 18)
HANDLE CHECK-AHEAD (FIG. 15)
BACKGROUND SWEEP COMPLETES EVENT
CONTINUE BACKGROUND SWEEP (FIG. 20)
BACKGROUND SWEEP ACTIVE?
YES
NO
WAIT FOR NEXT HOST COMMAND
BACKGROUND SWEEP TIMES OUT
INITATE BACKGROUND SWEEP (FIG.17)
WAIT FOR SWEEP EVENT TO COMPLETE

FIG. 4

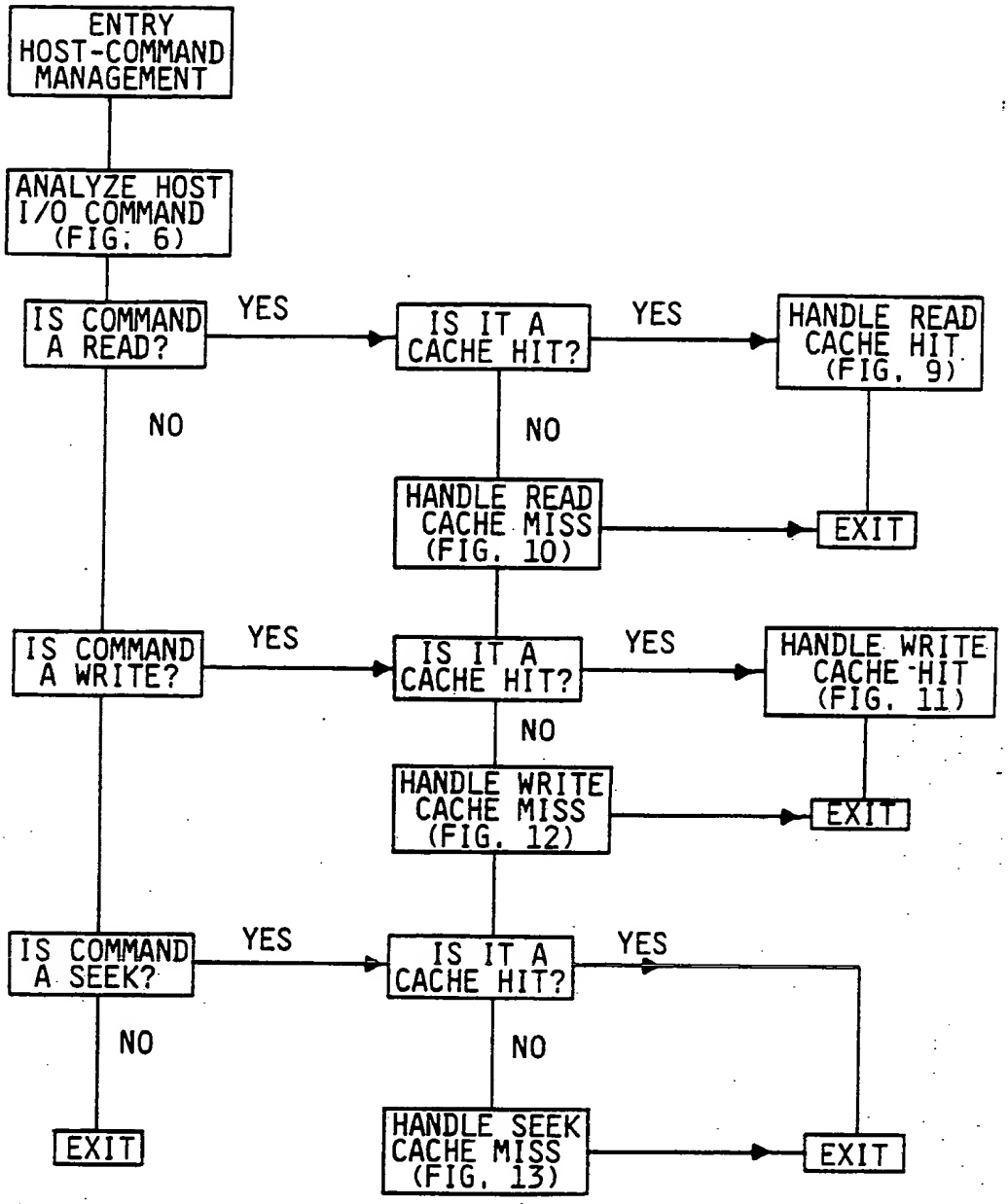

ENTRY HOST-COMMAND MANAGEMENT
ANALYZE HOST I/O COMMAND (FIG. 6)
IS COMMAND A READ?
YES
IS IT A CACHE HIT?
YES
HANDLE READ CACHE HIT (FIG. 9)
NO
NO
HANDLE READ CACHE MISS (FIG. 10)
EXIT
IS COMMAND A WRITE?
YES
IS IT A CACHE HIT?
YES
HANDLE WRITE CACHE HIT (FIG. 11)
NO
HANDLE WRITE CACHE MISS (FIG. 12)
EXIT
IS COMMAND A SEEK?
YES
IS IT A CACHE HIT?
YES
NO
NO
EXIT
HANDLE SEEK CACHE MISS (FIG. 13)
EXIT

FIG. 5

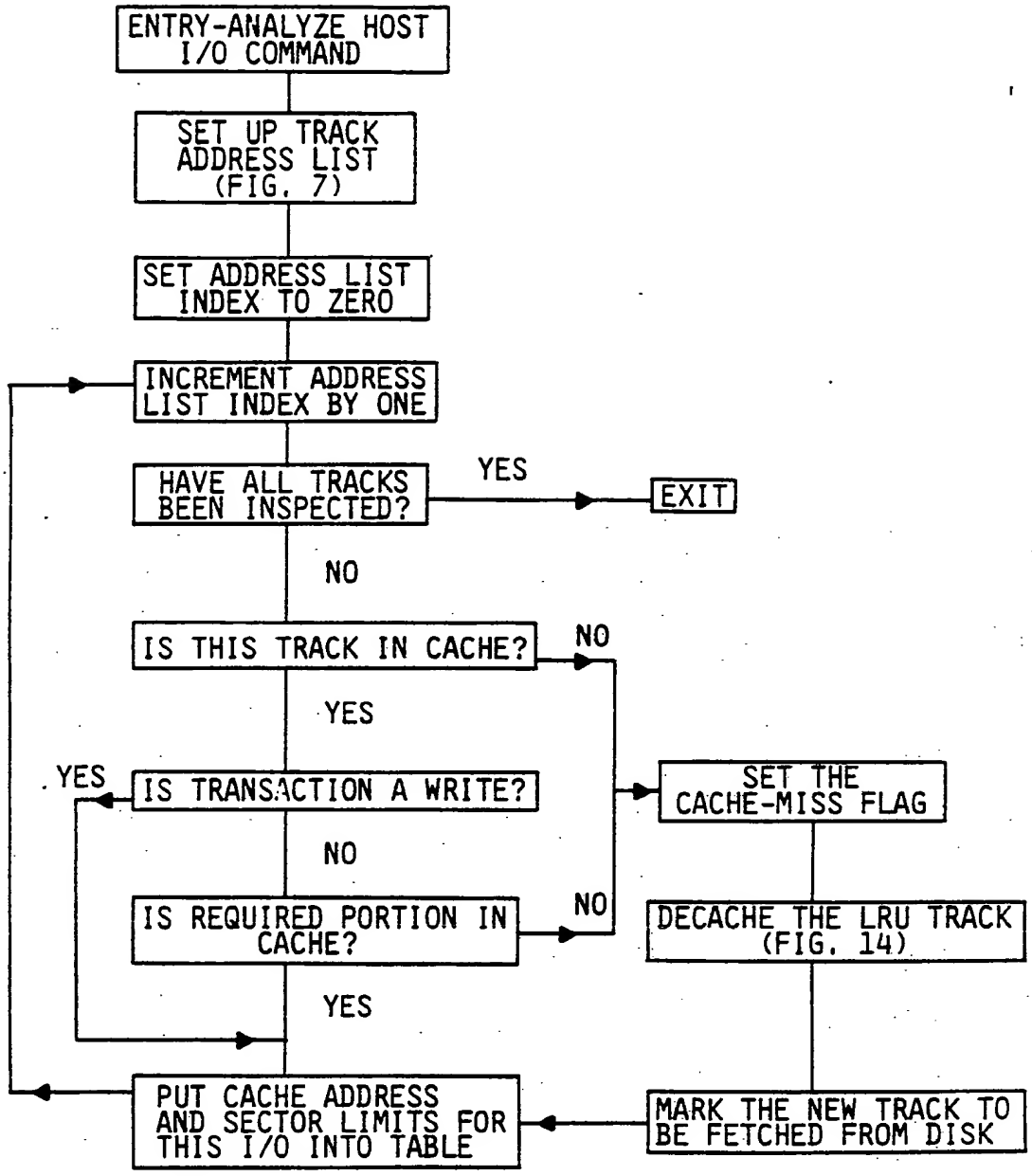

ENTRY-ANALYZE HOST I/O COMMAND
SET UP TRACK ADDRESS LIST (FIG. 7)
SET ADDRESS LIST INDEX TO ZERO
INCREMENT ADDRESS LIST INDEX BY ONE
HAVE ALL TRACKS BEEN INSPECTED?
YES
EXIT
NO
IS THIS TRACK IN CACHE?
NO
YES
YES
IS TRANSACTION A WRITE?
SET THE CACHE-MISS FLAG
NO
IS REQUIRED PORTION IN CACHE?
NO
DECACHE THE LRU TRACK (FIG. 14)
YES
PUT CACHE ADDRESS AND SECTOR LIMITS FOR THIS I/O INTO TABLE
MARK THE NEW TRACK TO BE FETCHED FROM DISK

FIG. 6

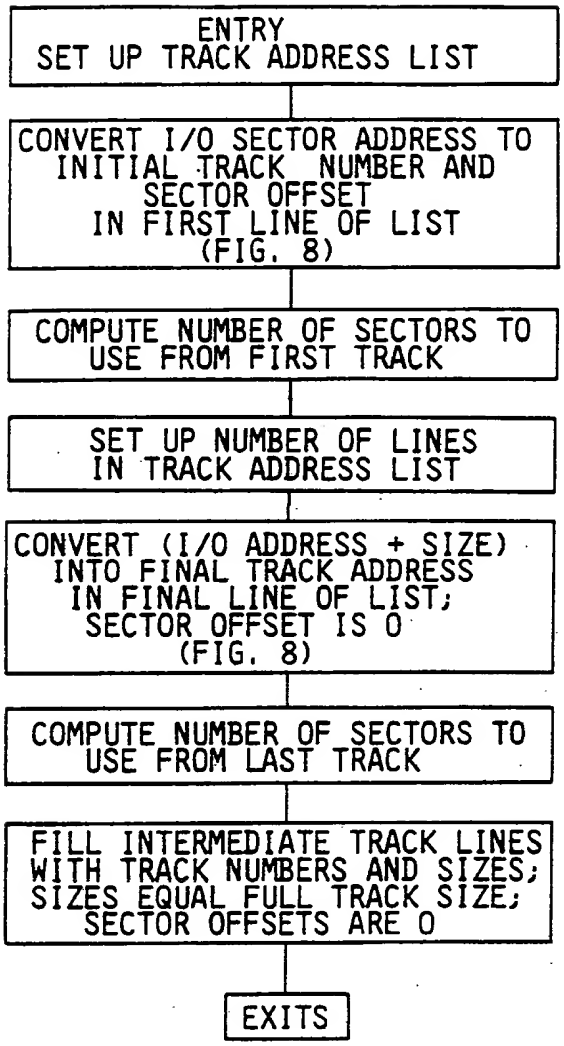

ENTRY
SET UP TRACK ADDRESS LIST
CONVERT I/O SECTOR ADDRESS TO INITIAL TRACK NUMBER AND SECTOR OFFSET IN FIRST LINE OF LIST (FIG. 8)
COMPUTE NUMBER OF SECTORS TO USE FROM FIRST TRACK
SET UP NUMBER OF LINES IN TRACK ADDRESS LIST
CONVERT (I/O ADDRESS + SIZE) INTO FINAL TRACK ADDRESS IN FINAL LINE OF LIST; SECTOR OFFSET IS 0 (FIG. 8)
COMPUTE NUMBER OF SECTORS TO USE FROM LAST TRACK
FILL INTERMEDIATE TRACK LINES WITH TRACK NUMBERS AND SIZES; SIZES EQUAL FULL TRACK SIZE; SECTOR OFFSETS ARE 0
EXITS

FIG. 7

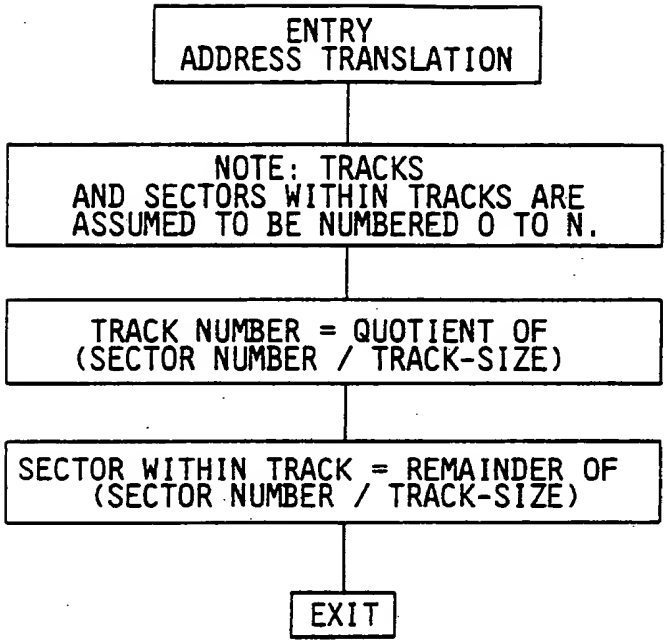

ENTRY
ADDRESS TRANSLATION
NOTE: TRACKS
AND SECTORS WITHIN TRACKS ARE
ASSUMED TO BE NUMBERED 0 TO N.
TRACK NUMBER = QUOTIENT OF
(SECTOR NUMBER / TRACK-SIZE)
SECTOR WITHIN TRACK = REMAINDER OF
(SECTOR NUMBER / TRACK-SIZE)
EXIT

FIG. 8

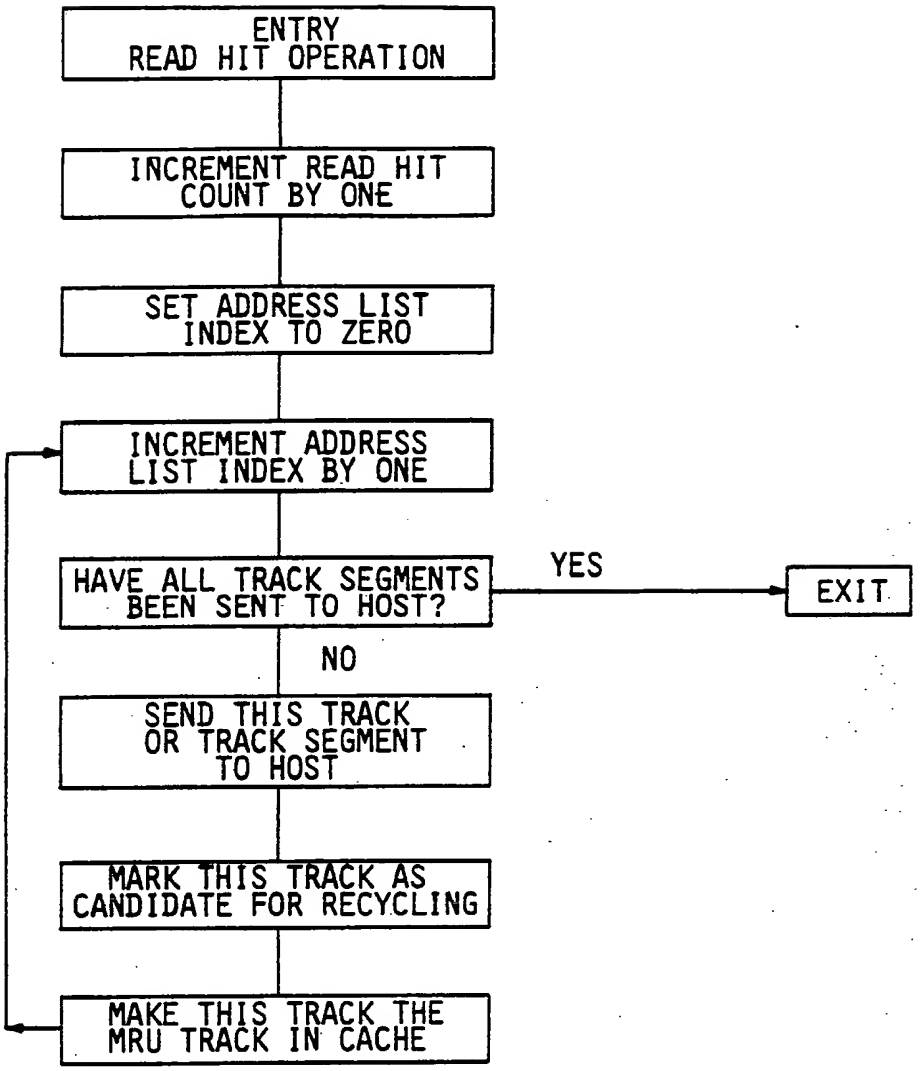

ENTRY
READ HIT OPERATION
INCREMENT READ HIT COUNT BY ONE
SET ADDRESS LIST INDEX TO ZERO
INCREMENT ADDRESS LIST INDEX BY ONE
HAVE ALL TRACK SEGMENTS BEEN SENT TO HOST?
YES
EXIT
NO
SEND THIS TRACK OR TRACK SEGMENT TO HOST
MARK THIS TRACK AS CANDIDATE FOR RECYCLING
MAKE THIS TRACK THE MRU TRACK IN CACHE

FIG. 9

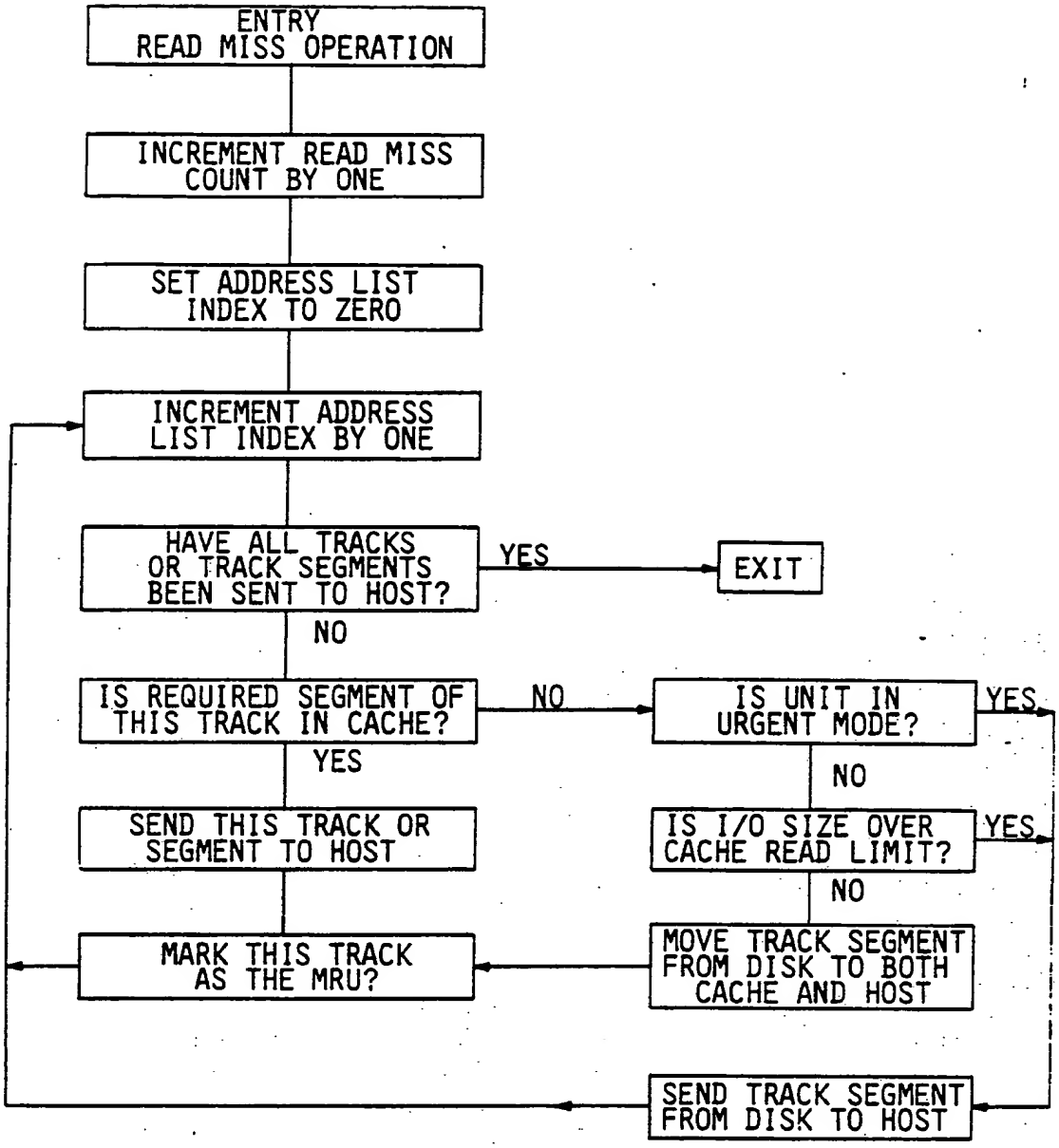

ENTRY
READ MISS OPERATION
INCREMENT READ MISS COUNT BY ONE
SET ADDRESS LIST INDEX TO ZERO
INCREMENT ADDRESS LIST INDEX BY ONE
HAVE ALL TRACKS OR TRACK SEGMENTS BEEN SENT TO HOST?
YES
EXIT
NO
IS REQUIRED SEGMENT OF THIS TRACK IN CACHE?
NO
IS UNIT IN URGENT MODE?
YES
YES
NO
SEND THIS TRACK OR SEGMENT TO HOST
IS I/O SIZE OVER CACHE READ LIMIT?
YES
NO
MARK THIS TRACK AS THE MRU?
MOVE TRACK SEGMENT FROM DISK TO BOTH CACHE AND HOST
SEND TRACK SEGMENT FROM DISK TO HOST

FIG. 10

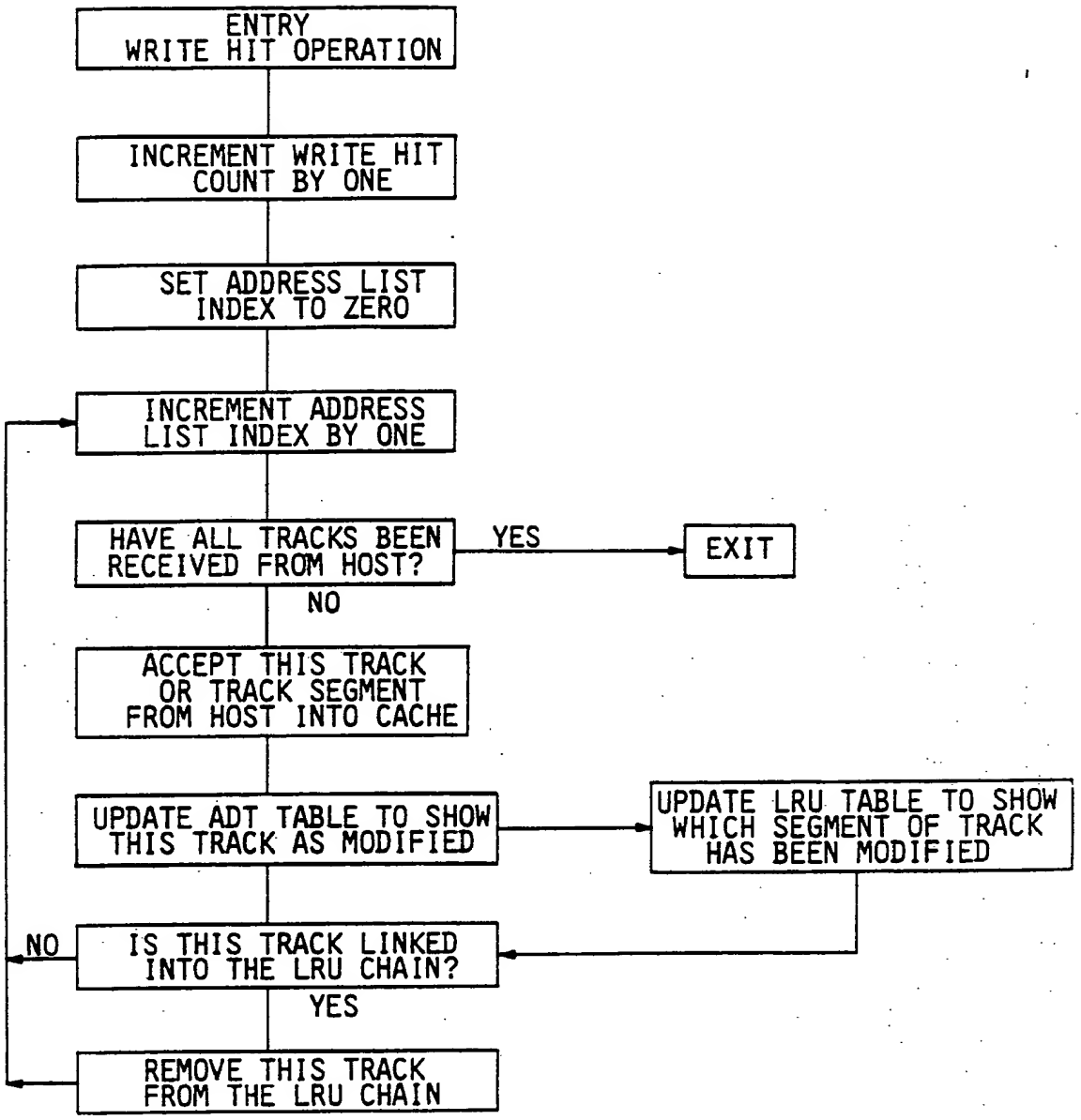

ENTRY
WRITE HIT OPERATION
INCREMENT WRITE HIT COUNT BY ONE
SET ADDRESS LIST INDEX TO ZERO
INCREMENT ADDRESS LIST INDEX BY ONE
HAVE ALL TRACKS BEEN RECEIVED FROM HOST?
YES
EXIT
NO
ACCEPT THIS TRACK OR TRACK SEGMENT FROM HOST INTO CACHE
UPDATE ADT TABLE TO SHOW THIS TRACK AS MODIFIED
UPDATE LRU TABLE TO SHOW WHICH SEGMENT OF TRACK HAS BEEN MODIFIED
NO
IS THIS TRACK LINKED INTO THE LRU CHAIN?
YES
REMOVE THIS TRACK FROM THE LRU CHAIN

FIG. 11

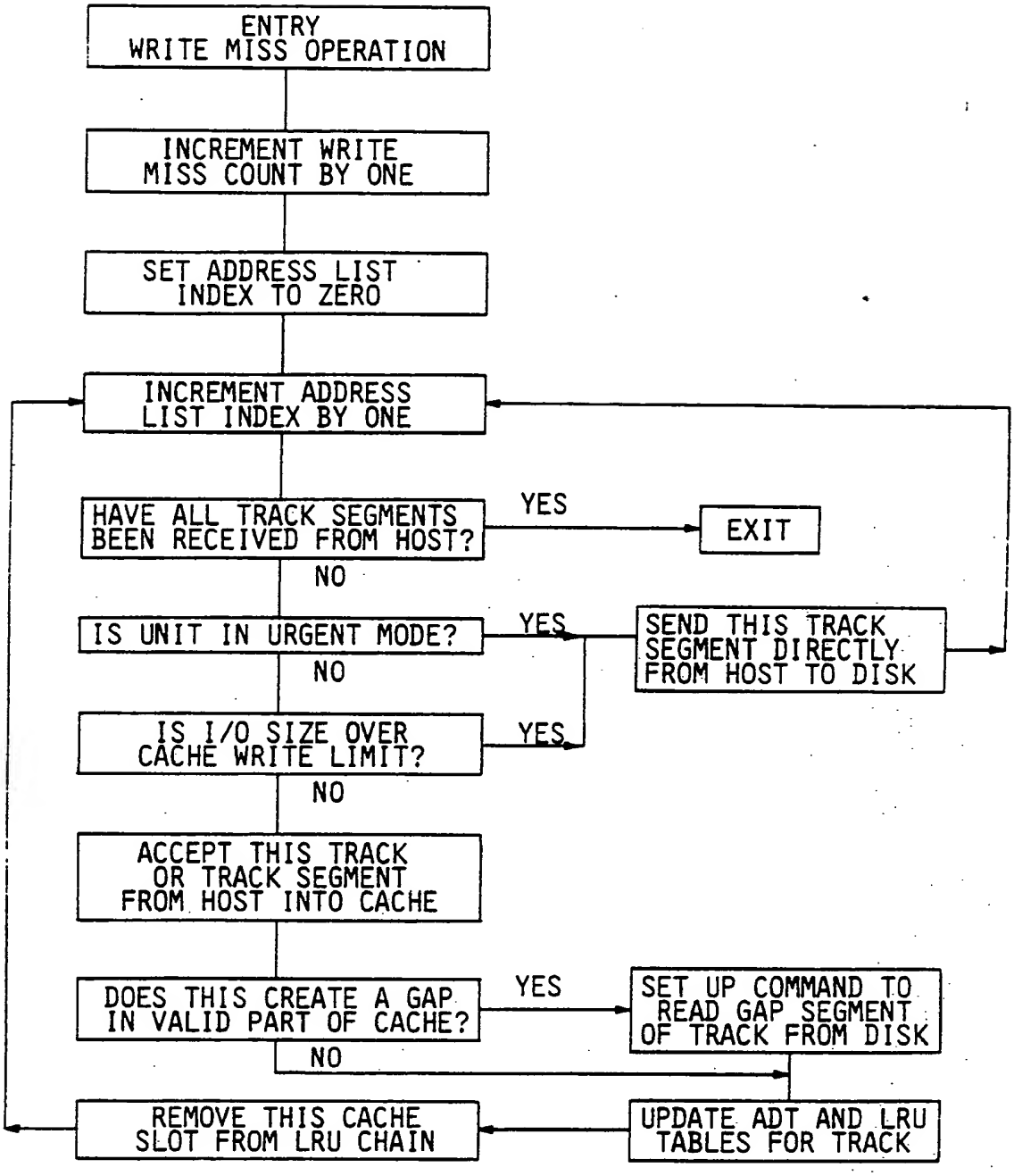

ENTRY
WRITE MISS OPERATION
INCREMENT WRITE
MISS COUNT BY ONE
SET ADDRESS LIST
INDEX TO ZERO
INCREMENT ADDRESS
LIST INDEX BY ONE
HAVE ALL TRACK SEGMENTS
BEEN RECEIVED FROM HOST?
YES
EXIT
NO
IS UNIT IN URGENT MODE?
YES
SEND THIS TRACK
SEGMENT DIRECTLY
FROM HOST TO DISK
NO
IS I/O SIZE OVER
CACHE WRITE LIMIT?
YES
NO
ACCEPT THIS TRACK
OR TRACK SEGMENT
FROM HOST INTO CACHE
DOES THIS CREATE A GAP
IN VALID PART OF CACHE?
YES
SET UP COMMAND TO
READ GAP SEGMENT
OF TRACK FROM DISK
NO
REMOVE THIS CACHE
SLOT FROM LRU CHAIN
UPDATE ADT AND LRU
TABLES FOR TRACK

FIG. 12

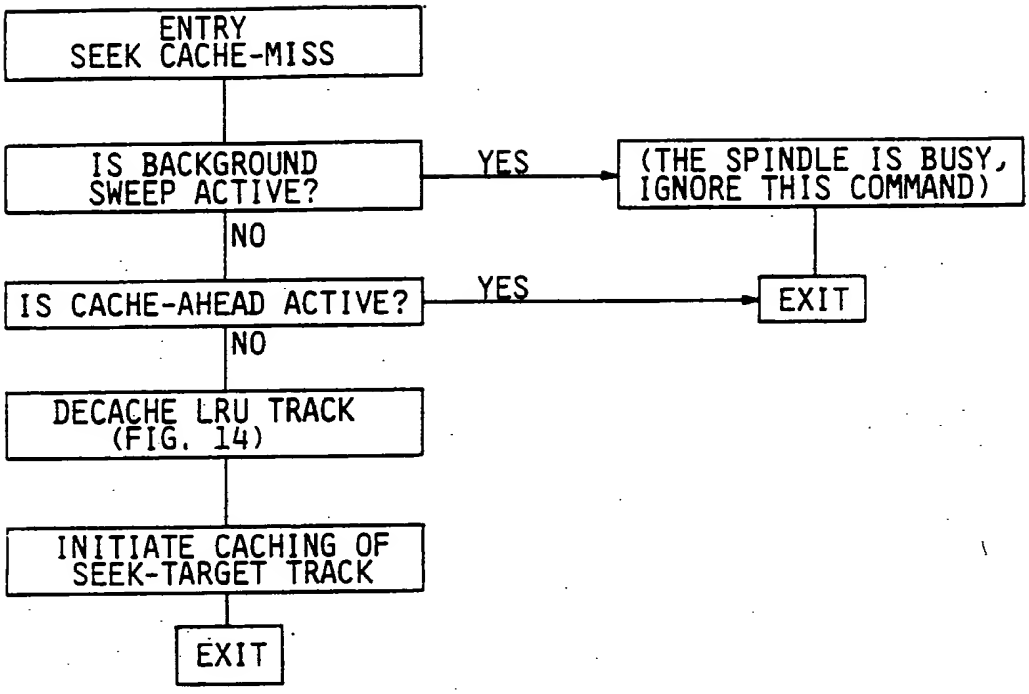

ENTRY
SEEK CACHE-MISS
IS BACKGROUND SWEEP ACTIVE?
YES
(THE SPINDLE IS BUSY, IGNORE THIS COMMAND)
NO
IS CACHE-AHEAD ACTIVE?
YES
EXIT
NO
DECACHE LRU TRACK (FIG. 14)
INITIATE CACHING OF SEEK-TARGET TRACK
EXIT

FIG. 13

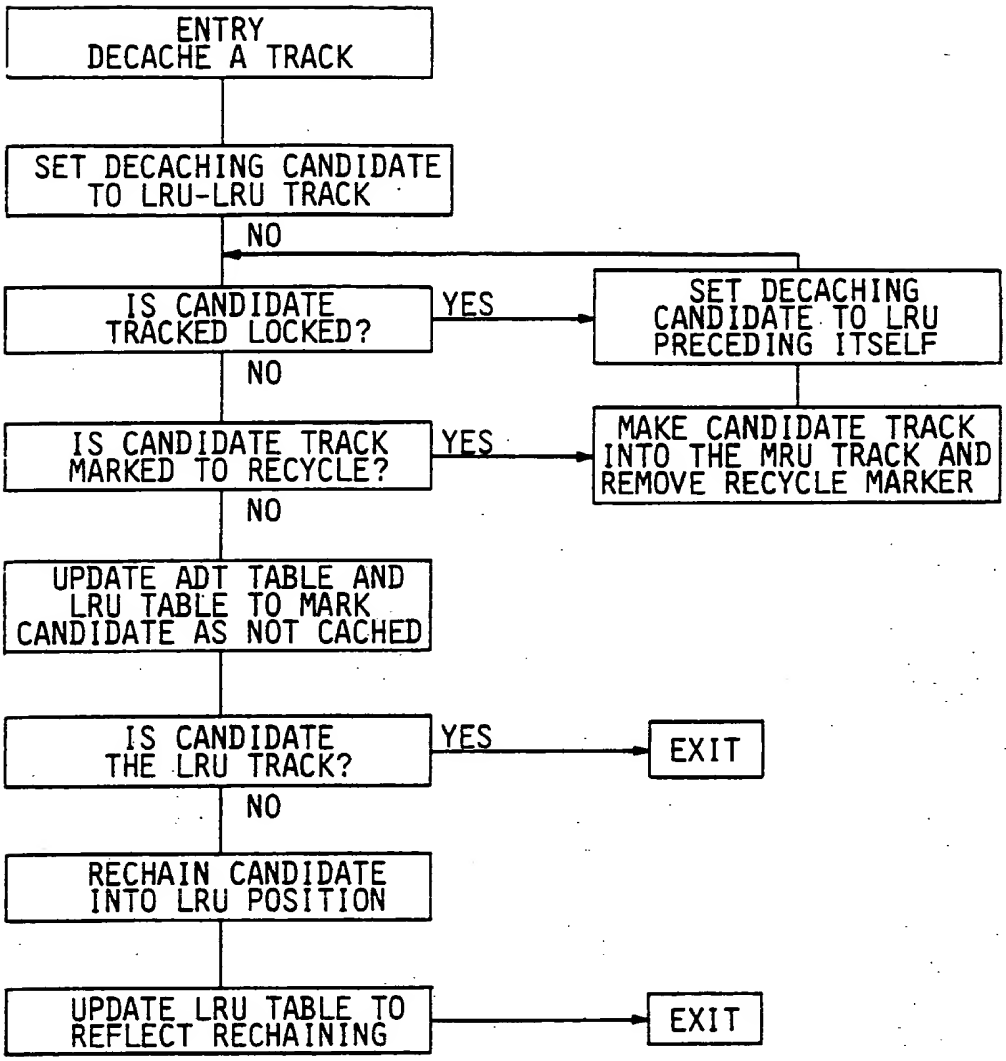

ENTRY
DECACHE A TRACK
SET DECACHING CANDIDATE TO LRU-LRU TRACK
NO
IS CANDIDATE TRACKED LOCKED?
YES
SET DECACHING CANDIDATE TO LRU PRECEDING ITSELF
NO
IS CANDIDATE TRACK MARKED TO RECYCLE?
YES
MAKE CANDIDATE TRACK INTO THE MRU TRACK AND REMOVE RECYCLE MARKER
NO
UPDATE ADT TABLE AND LRU TABLE TO MARK CANDIDATE AS NOT CACHED
IS CANDIDATE THE LRU TRACK?
YES
EXIT
NO
RECHAIN CANDIDATE INTO LRU POSITION
UPDATE LRU TABLE TO REFLECT RECHAINING
EXIT

FIG. 14

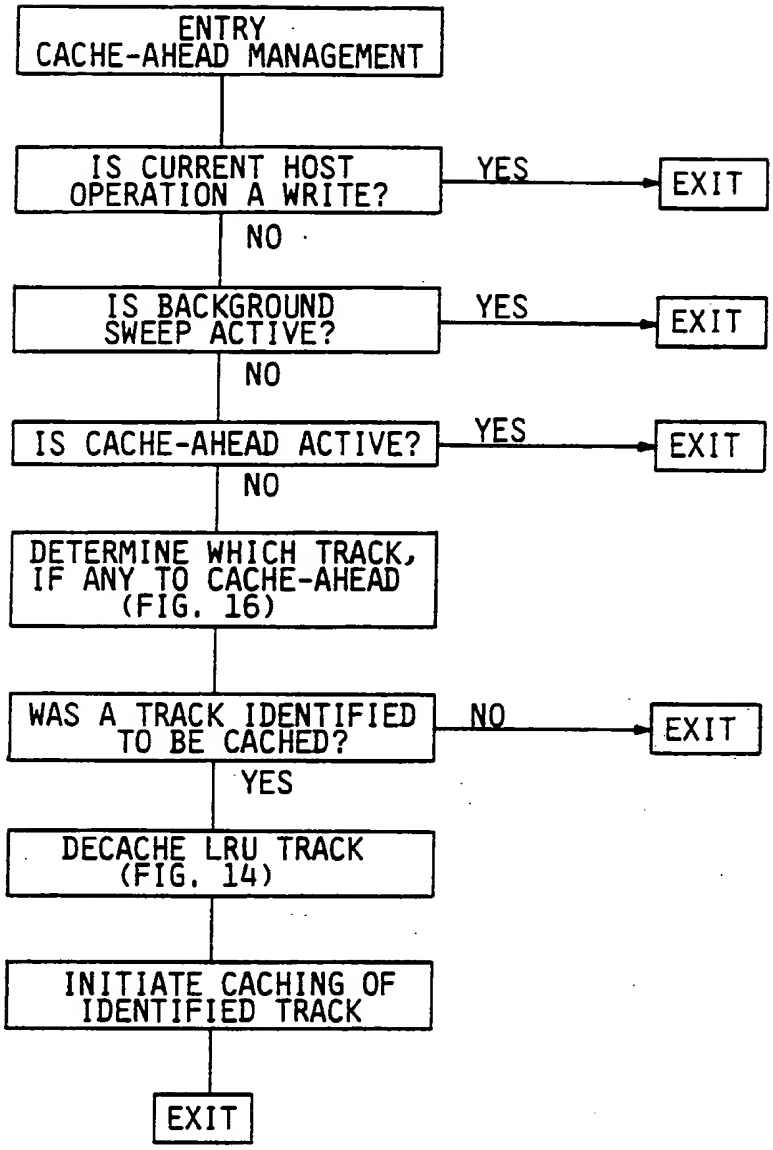

ENTRY
CACHE-AHEAD MANAGEMENT
IS CURRENT HOST OPERATION A WRITE?
YES
EXIT
NO
IS BACKGROUND SWEEP ACTIVE?
YES
EXIT
NO
IS CACHE-AHEAD ACTIVE?
YES
EXIT
NO
DETERMINE WHICH TRACK, IF ANY TO CACHE-AHEAD (FIG. 16)
WAS A TRACK IDENTIFIED TO BE CACHED?
NO
EXIT
YES
DECACHE LRU TRACK (FIG. 14)
INITIATE CACHING OF IDENTIFIED TRACK
EXIT

FIG. 15

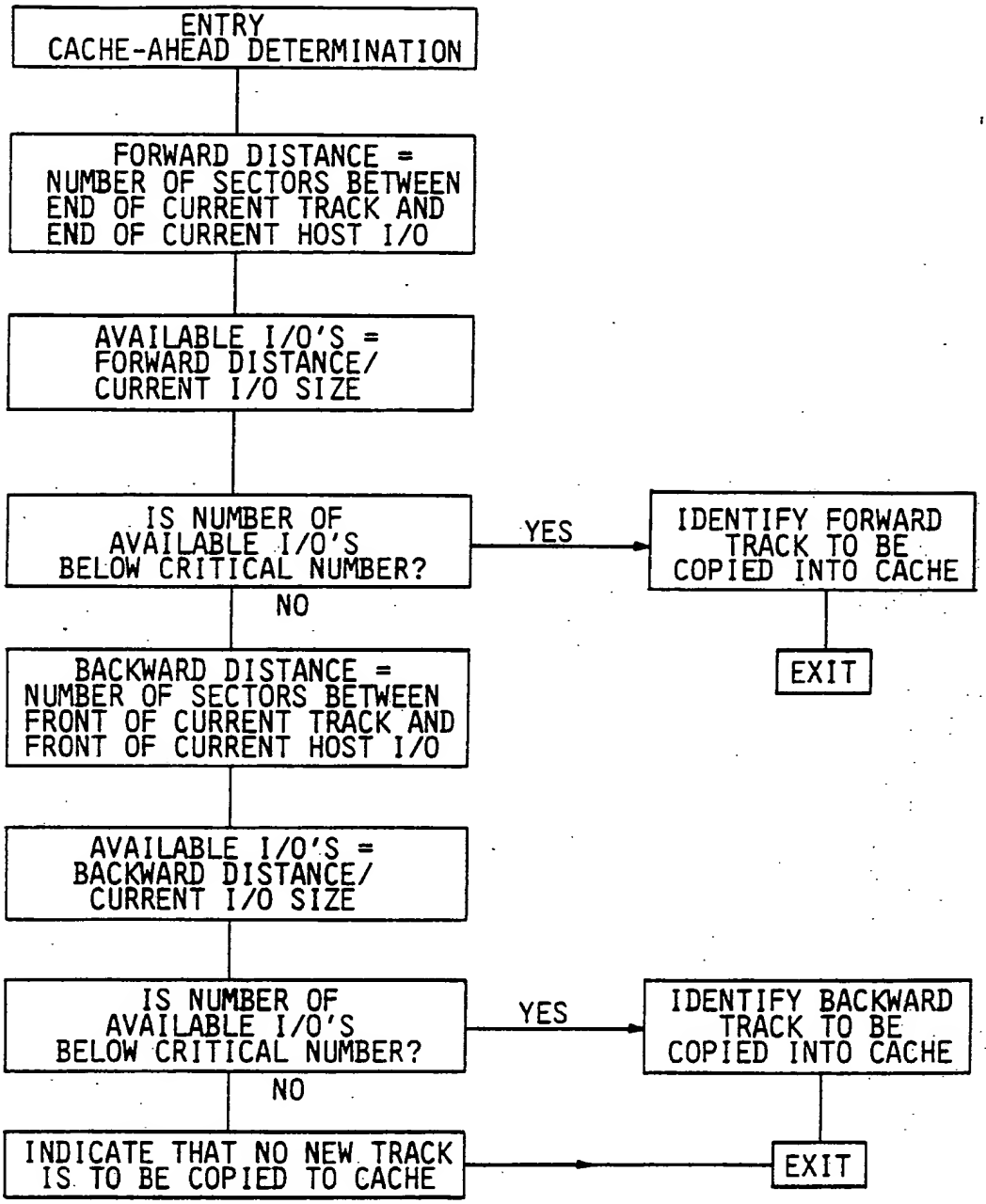

ENTRY
CACHE-AHEAD DETERMINATION
FORWARD DISTANCE = NUMBER OF SECTORS BETWEEN END OF CURRENT TRACK AND END OF CURRENT HOST I/O
AVAILABLE I/O'S = FORWARD DISTANCE/ CURRENT I/O SIZE
IS NUMBER OF AVAILABLE I/O'S BELOW CRITICAL NUMBER?
YES
IDENTIFY FORWARD TRACK TO BE COPIED INTO CACHE
EXIT
NO
BACKWARD DISTANCE = NUMBER OF SECTORS BETWEEN FRONT OF CURRENT TRACK AND FRONT OF CURRENT HOST I/O
AVAILABLE I/O'S = BACKWARD DISTANCE/ CURRENT I/O SIZE
IS NUMBER OF AVAILABLE I/O'S BELOW CRITICAL NUMBER?
YES
IDENTIFY BACKWARD TRACK TO BE COPIED INTO CACHE
NO
INDICATE THAT NO NEW TRACK IS TO BE COPIED TO CACHE
EXIT

FIG. 16

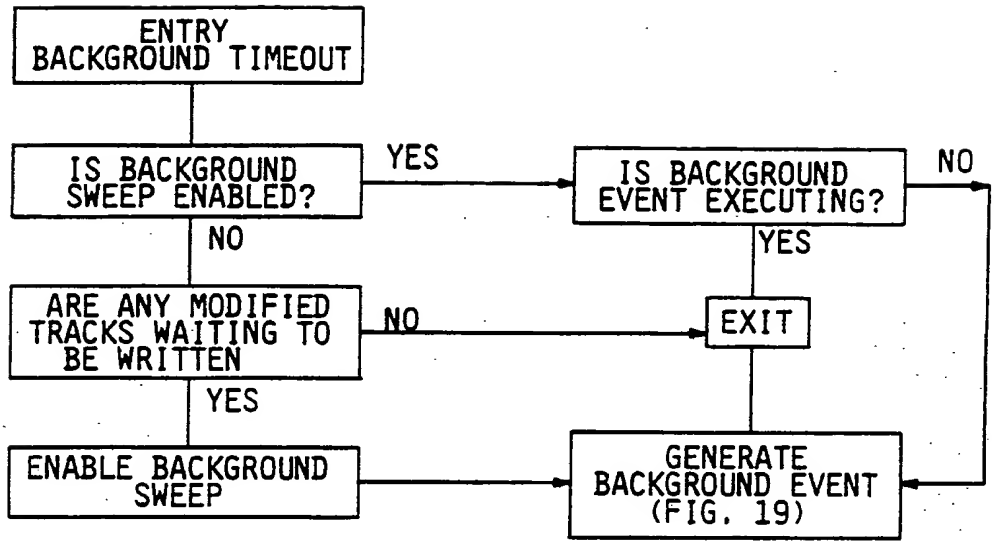

ENTRY
BACKGROUND TIMEOUT
IS BACKGROUND SWEEP ENABLED?
YES
IS BACKGROUND EVENT EXECUTING?
NO
NO
YES
ARE ANY MODIFIED TRACKS WAITING TO BE WRITTEN
NO
EXIT
YES
ENABLE BACKGROUND SWEEP
GENERATE BACKGROUND EVENT (FIG. 19)

FIG. 17

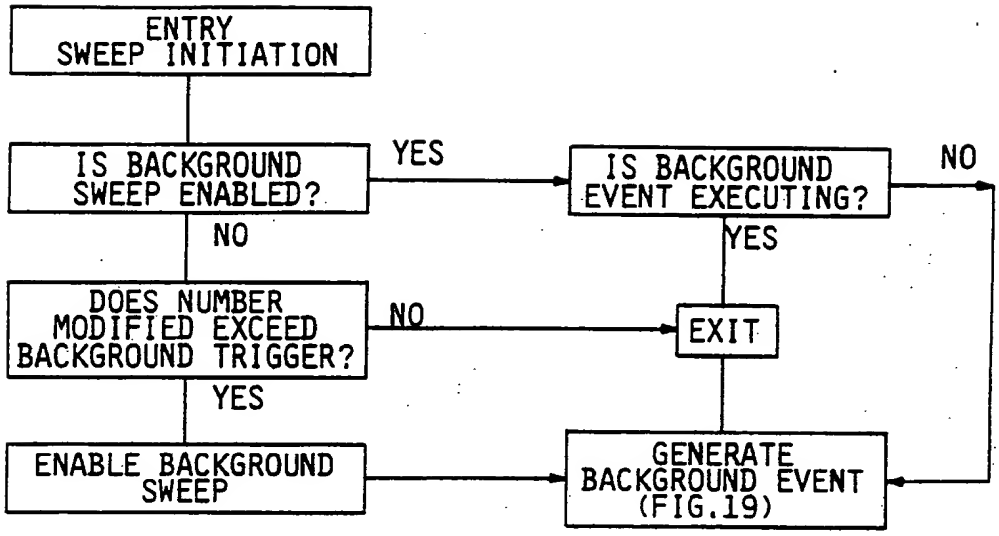

ENTRY
SWEEP INITIATION
IS BACKGROUND SWEEP ENABLED?
YES
IS BACKGROUND EVENT EXECUTING?
NO
NO
YES
DOES NUMBER MODIFIED EXCEED BACKGROUND TRIGGER?
NO
EXIT
YES
ENABLE BACKGROUND SWEEP
GENERATE BACKGROUND EVENT (FIG.19)

FIG. 18

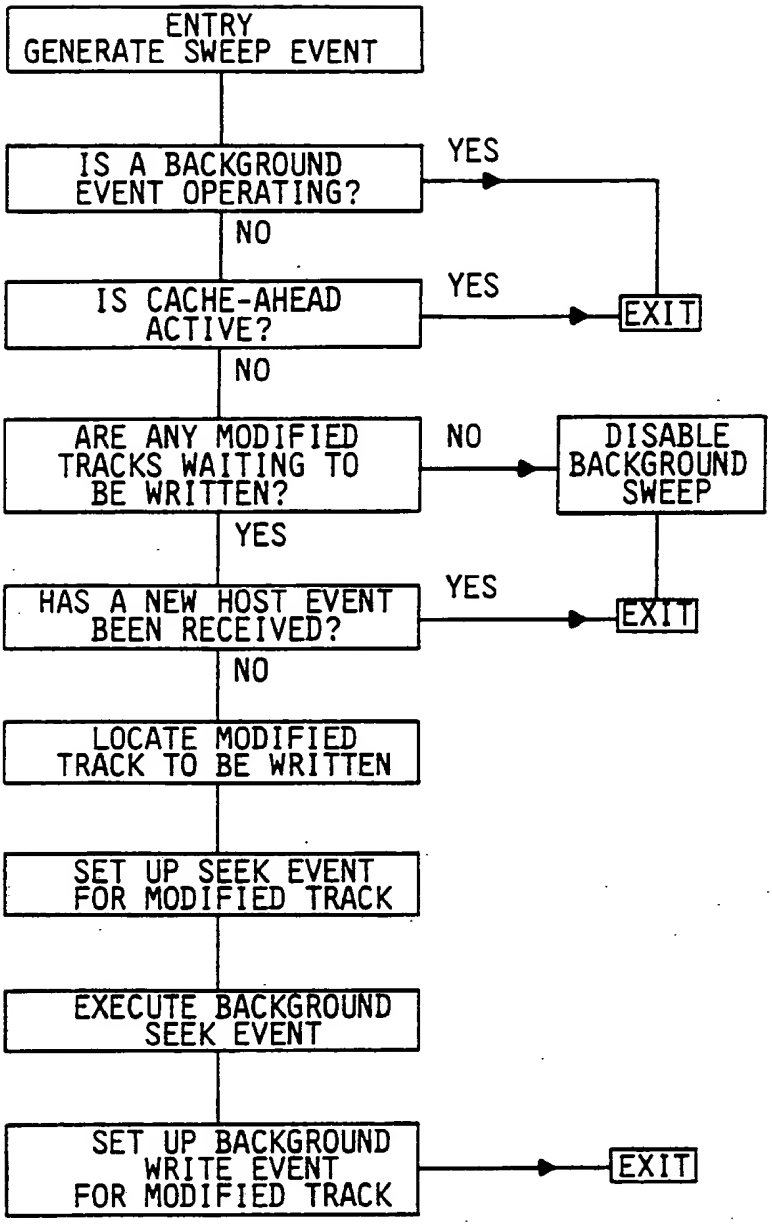

ENTRY
GENERATE SWEEP EVENT
IS A BACKGROUND EVENT OPERATING?
YES
NO
IS CACHE-AHEAD ACTIVE?
YES
EXIT
NO
ARE ANY MODIFIED TRACKS WAITING TO BE WRITTEN?
NO
DISABLE BACKGROUND SWEEP
YES
HAS A NEW HOST EVENT BEEN RECEIVED?
YES
EXIT
NO
LOCATE MODIFIED TRACK TO BE WRITTEN
SET UP SEEK EVENT FOR MODIFIED TRACK
EXECUTE BACKGROUND SEEK EVENT
SET UP BACKGROUND WRITE EVENT FOR MODIFIED TRACK
EXIT

FIG. 19

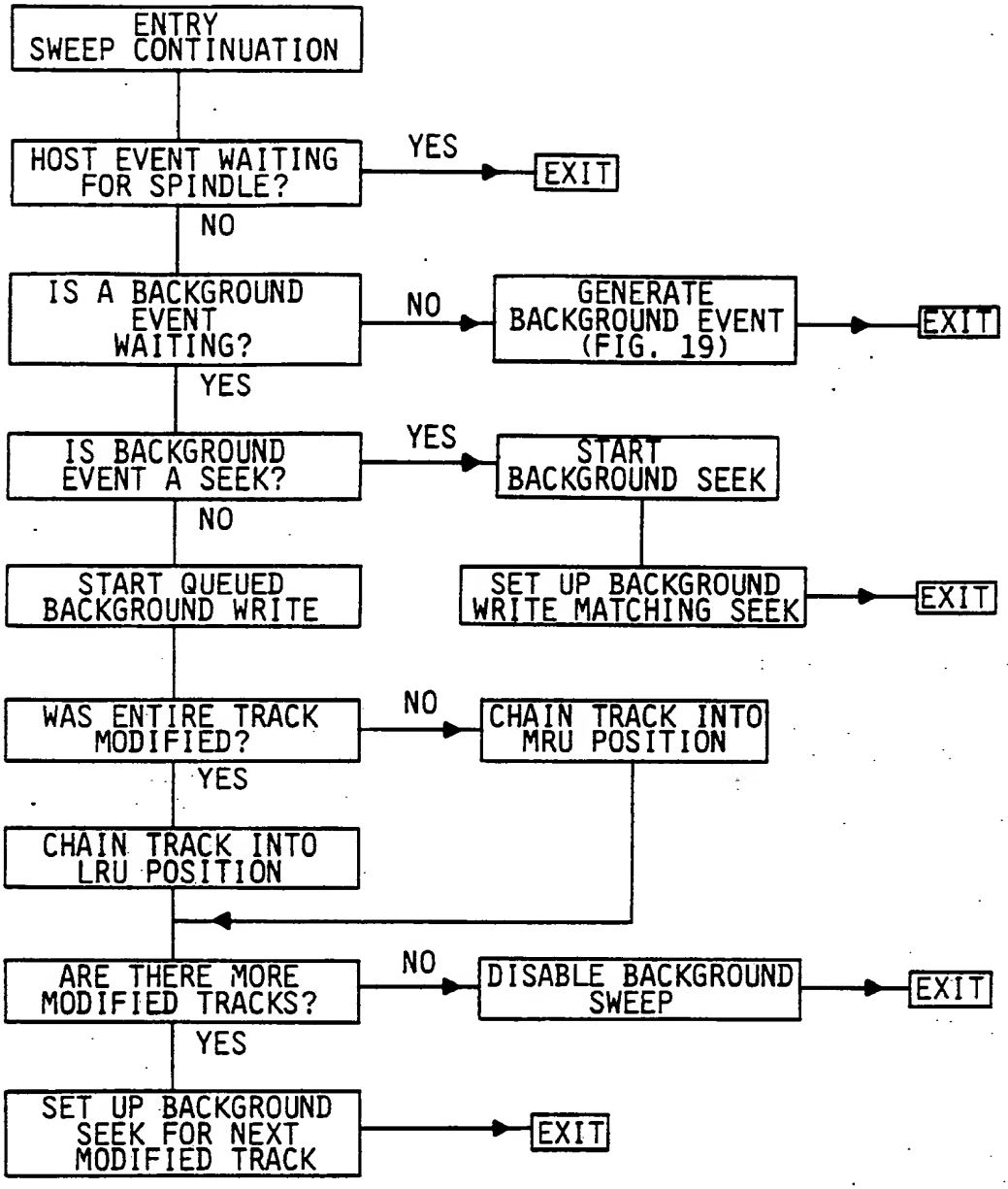

ENTRY
SWEEP CONTINUATION
HOST EVENT WAITING FOR SPINDLE?
YES
EXIT
NO
IS A BACKGROUND EVENT WAITING?
NO
GENERATE BACKGROUND EVENT (FIG. 19)
EXIT
YES
IS BACKGROUND EVENT A SEEK?
YES
START BACKGROUND SEEK
NO
START QUEUED BACKGROUND WRITE
SET UP BACKGROUND WRITE MATCHING SEEK
EXIT
WAS ENTIRE TRACK MODIFIED?
NO
CHAIN TRACK INTO MRU POSITION
YES
CHAIN TRACK INTO LRU POSITION
ARE THERE MORE MODIFIED TRACKS?
NO
DISABLE BACKGROUND SWEEP
EXIT
YES
SET UP BACKGROUND SEEK FOR NEXT MODIFIED TRACK
EXIT

FIG. 20

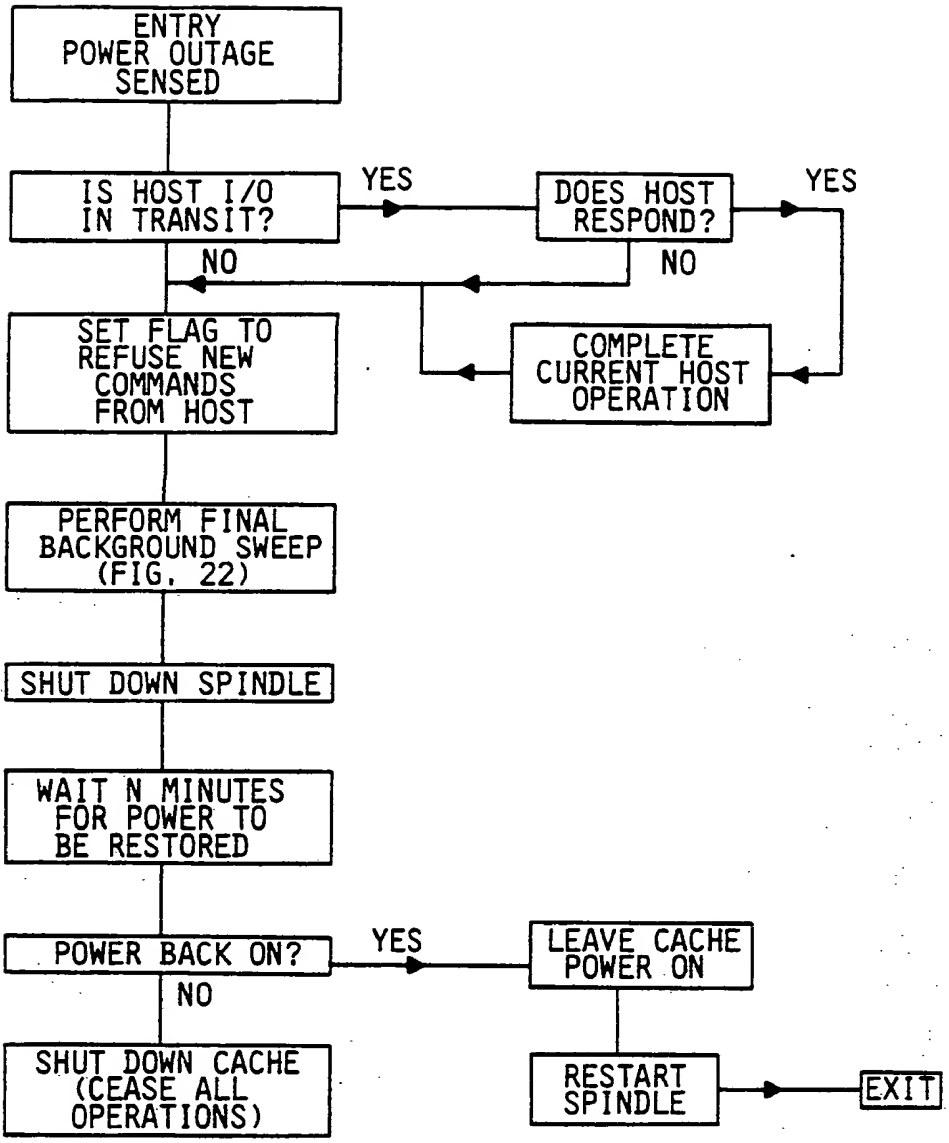

ENTRY
POWER OUTAGE SENSED
IS HOST I/O IN TRANSIT?
YES
DOES HOST RESPOND?
YES
NO
NO
SET FLAG TO REFUSE NEW COMMANDS FROM HOST
COMPLETE CURRENT HOST OPERATION
PERFORM FINAL BACKGROUND SWEEP (FIG. 22)
SHUT DOWN SPINDLE
WAIT N MINUTES FOR POWER TO BE RESTORED
POWER BACK ON?
YES
LEAVE CACHE POWER ON
NO
SHUT DOWN CACHE (CEASE ALL OPERATIONS)
RESTART SPINDLE
EXIT

FIG. 21

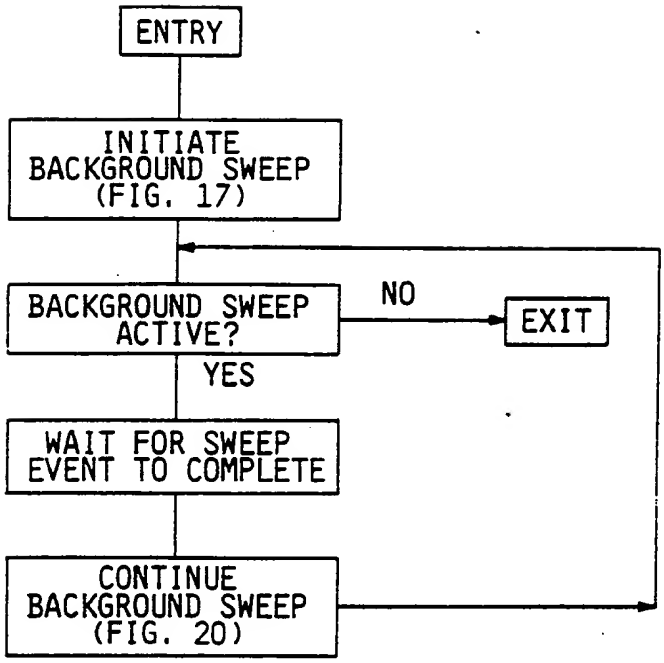

ENTRY
INITIATE BACKGROUND SWEEP (FIG. 17)
BACKGROUND SWEEP ACTIVE?
NO
EXIT
YES
WAIT FOR SWEEP EVENT TO COMPLETE
CONTINUE BACKGROUND SWEEP (FIG. 20)

FIG. 22